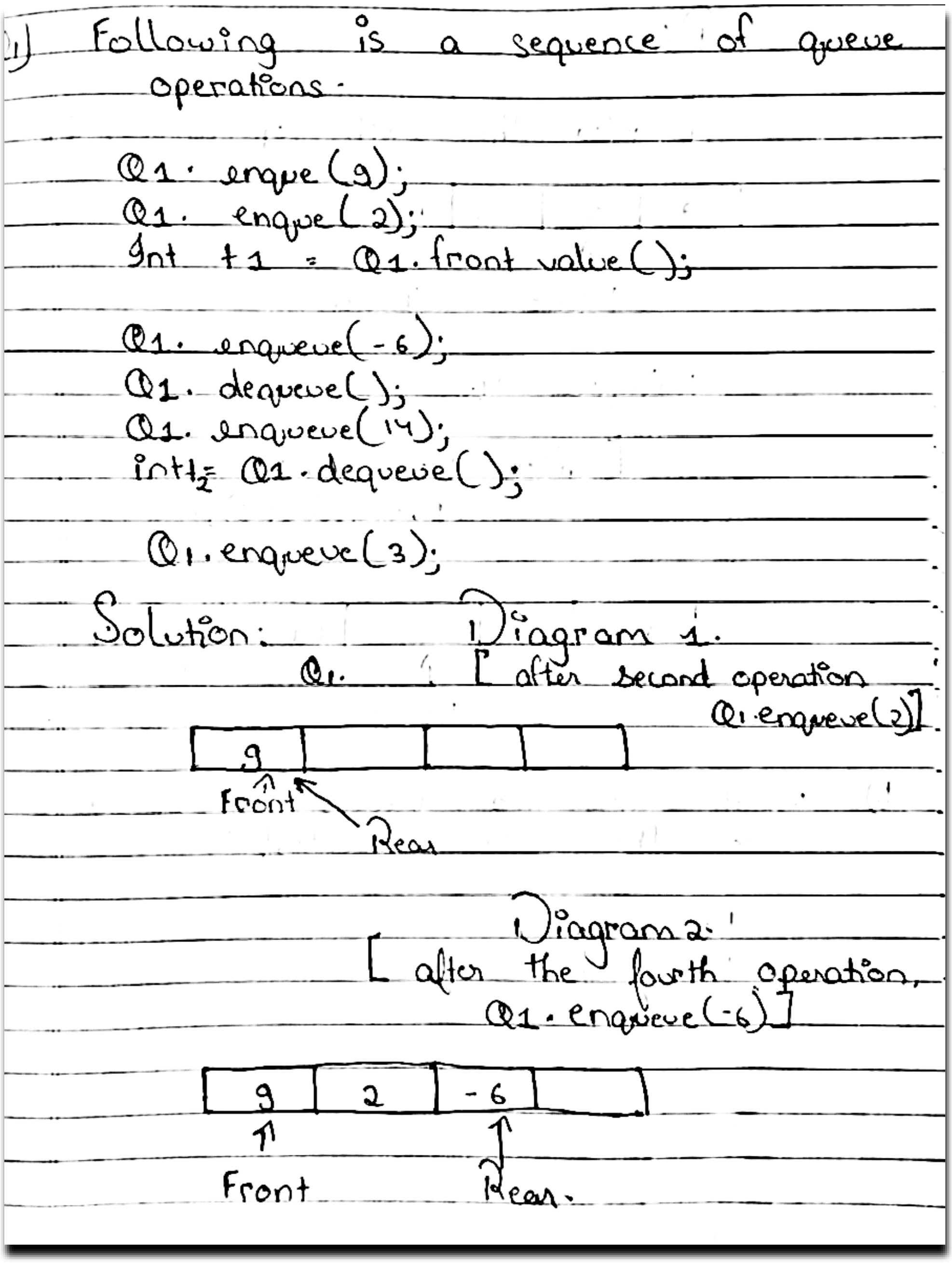
1. Following is a sequence of queue operations-

a)Assuming all instructions execute in the given sequence, draw four diagrams, showing the contents of the queue, after executing the second, fourth, sixth, and eighth In each diagram, include the values of all elements in the queue, and two pointers denoting the current "front" and "rear" of the queue.



1) Following is a sequence of queue operations.

```
Q1. enque(9);
Q1. enque(2);
Int t1 = Q1.front value();

Q1. enqueue(-6);
Q1. dequeue();
Q1. enqueue(14);
int t2= Q1.dequeue();

Q1.enqueue(3);
```

Solution:

Diagram 1.

Q1. [after second operation Q1.enqueue(2)]

9			

Front
Rear

Diagram 2:

[after the fourth operation, Q1.enqueue(-6)]

9	2	-6	

Front
Rear.

Diagram 3:

[after the sixth operation, Q1.enqueue(

9	2	-6	14
	↑		↑
	Front		Rear.

Diagram 4:

[after the eighth operation, Q1.enqueue(3)].

9	2	-6	14	3
		↑		↑
		Front		Rear.

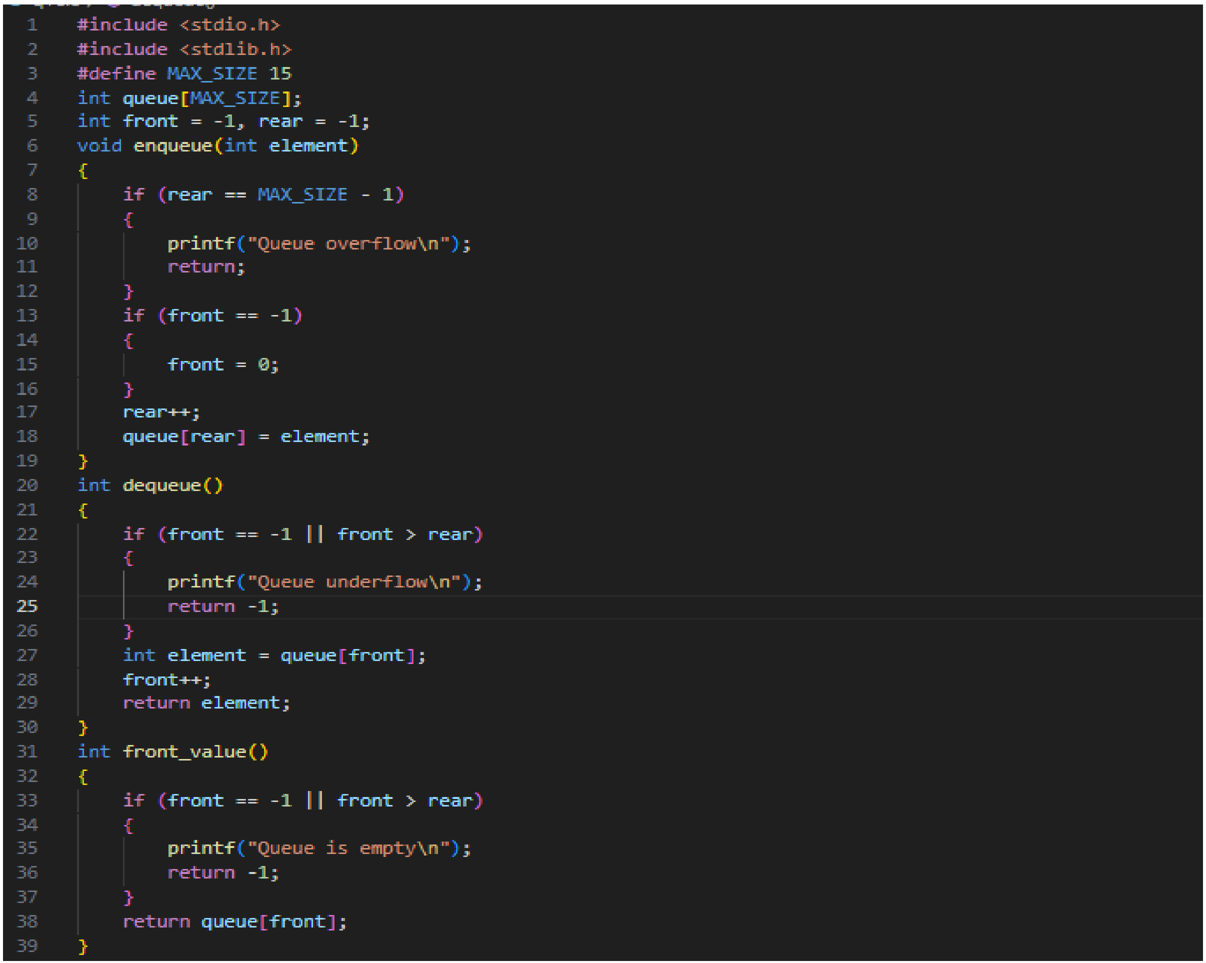
b) What are the values of t1and t2 after the code executes?

b) what are the values of T_1 and t_2 after the code executes?

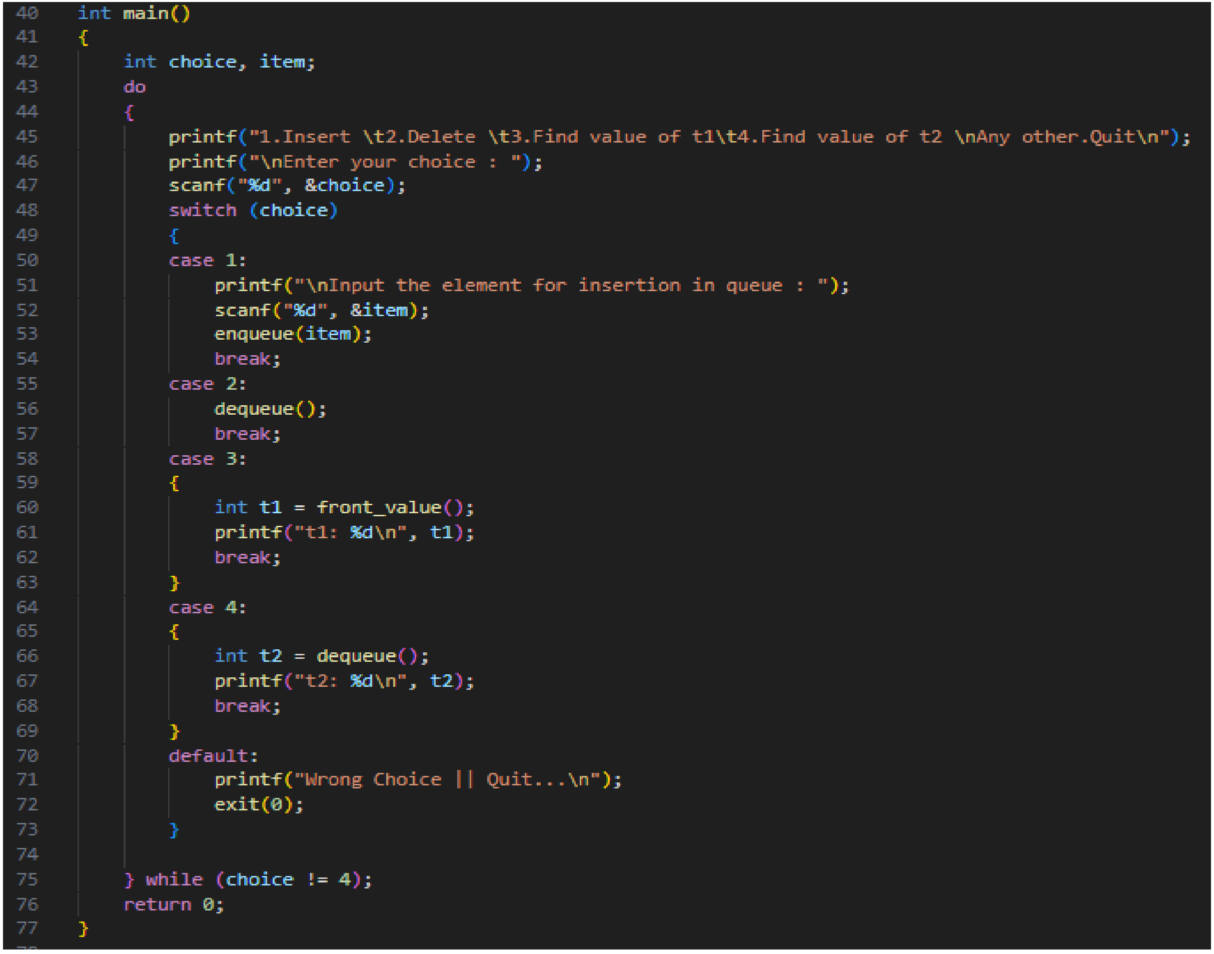
t1 = 9.

t2 = 2.

c) Finally write a program code(C) that would implement the same sequence of additions and modifications to a queue.

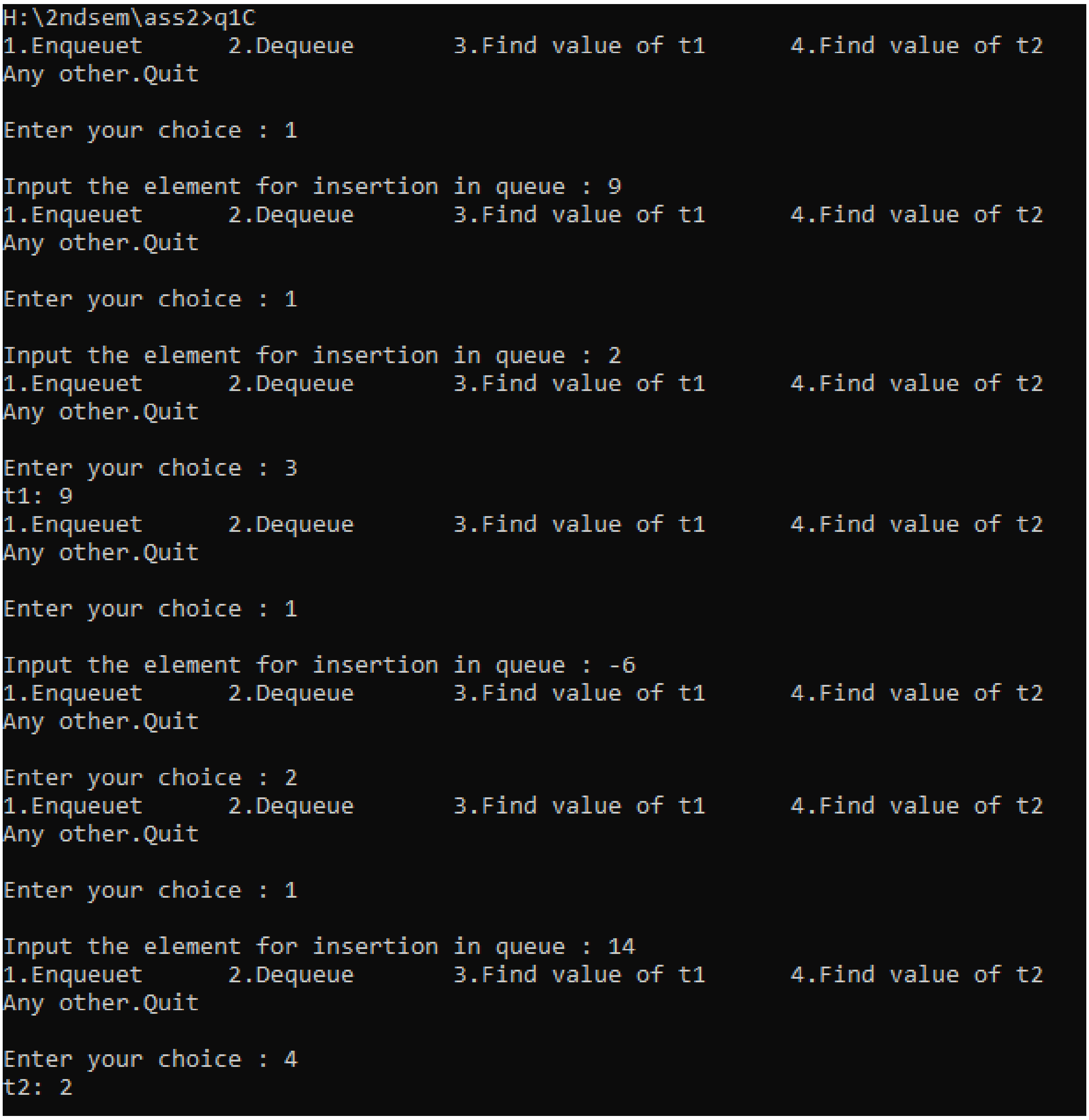


```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 15
int queue[MAX_SIZE];
int front = -1, rear = -1;
void enqueue(int element)
{
    if (rear == MAX_SIZE - 1)
    {
        printf("Queue overflow\n");
        return;
    }
    if (front == -1)
    {
        front = 0;
    }
    rear++;
    queue[rear] = element;
}
int dequeue()
{
    if (front == -1 || front > rear)
    {
        printf("Queue underflow\n");
        return -1;
    }
    int element = queue[front];
    front++;
    return element;
}
int front_value()
{
    if (front == -1 || front > rear)
    {
        printf("Queue is empty\n");
        return -1;
    }
    return queue[front];
}
```



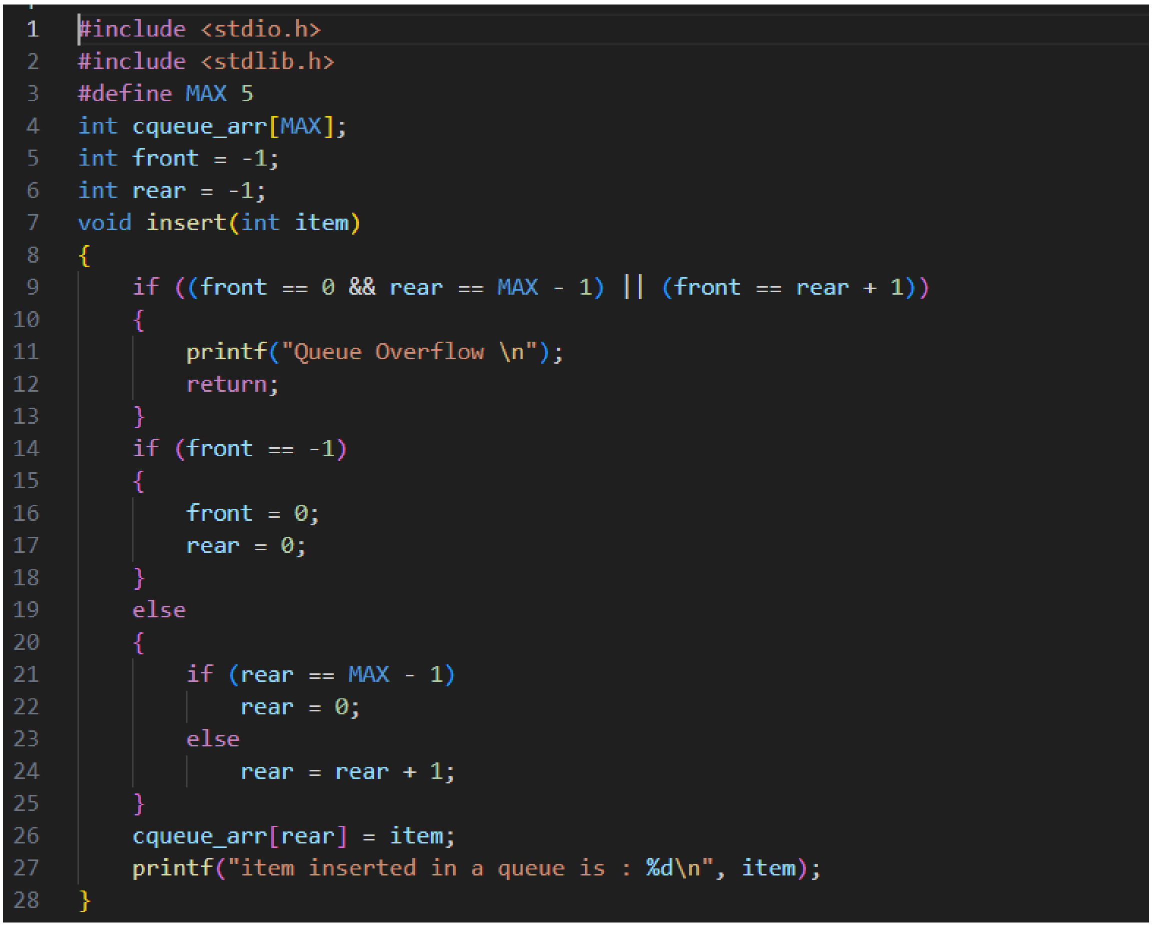
```c
int main()
{
    int choice, item;
    do
    {
        printf("1.Insert \t2.Delete \t3.Find value of t1\t4.Find value of t2 \nAny other.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nInput the element for insertion in queue : ");
            scanf("%d", &item);
            enqueue(item);
            break;
        case 2:
            dequeue();
            break;
        case 3:
        {
            int t1 = front_value();
            printf("t1: %d\n", t1);
            break;
        }
        case 4:
        {
            int t2 = dequeue();
            printf("t2: %d\n", t2);
            break;
        }
        default:
            printf("Wrong Choice || Quit...\n");
            exit(0);
        }

    } while (choice != 4);
    return 0;
}
```

Output



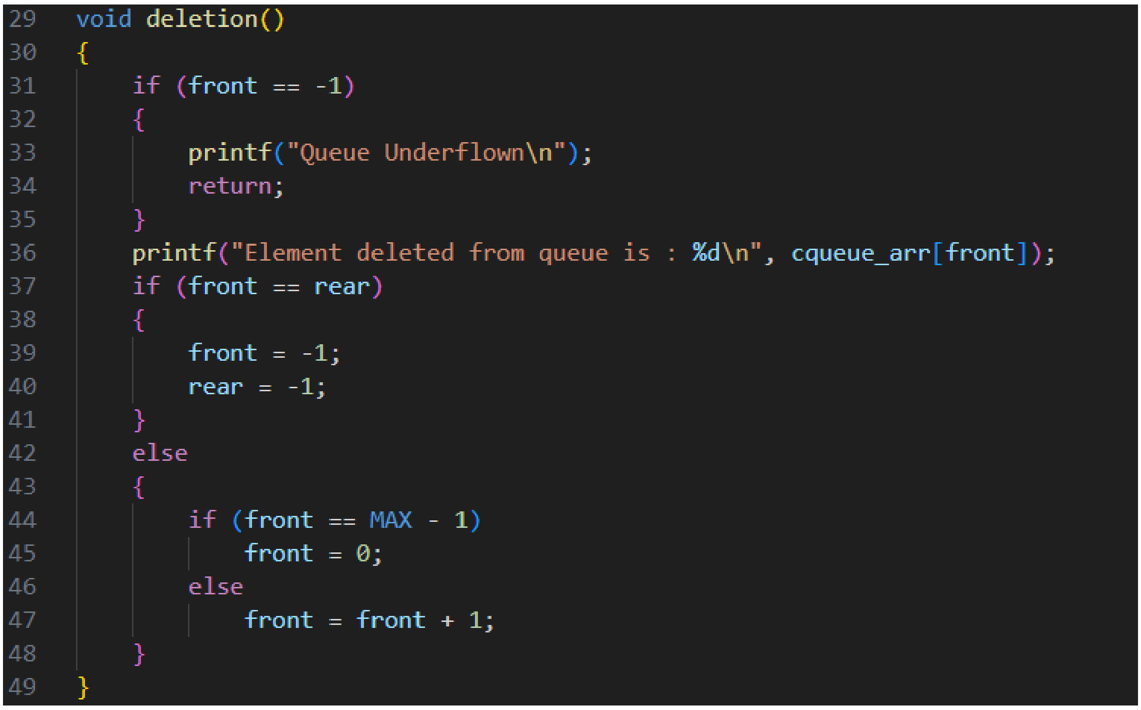
```
H:\2ndsem\ass2>q1C
1.Enqueuet      2.Dequeue       3.Find value of t1      4.Find value of t2
Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 9
1.Enqueuet      2.Dequeue       3.Find value of t1      4.Find value of t2
Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 2
1.Enqueuet      2.Dequeue       3.Find value of t1      4.Find value of t2
Any other.Quit

Enter your choice : 3
t1: 9
1.Enqueuet      2.Dequeue       3.Find value of t1      4.Find value of t2
Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : -6
1.Enqueuet      2.Dequeue       3.Find value of t1      4.Find value of t2
Any other.Quit

Enter your choice : 2
1.Enqueuet      2.Dequeue       3.Find value of t1      4.Find value of t2
Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 14
1.Enqueuet      2.Dequeue       3.Find value of t1      4.Find value of t2
Any other.Quit

Enter your choice : 4
t2: 2
```

2. Write a program using functions for implementation of circular Queue.



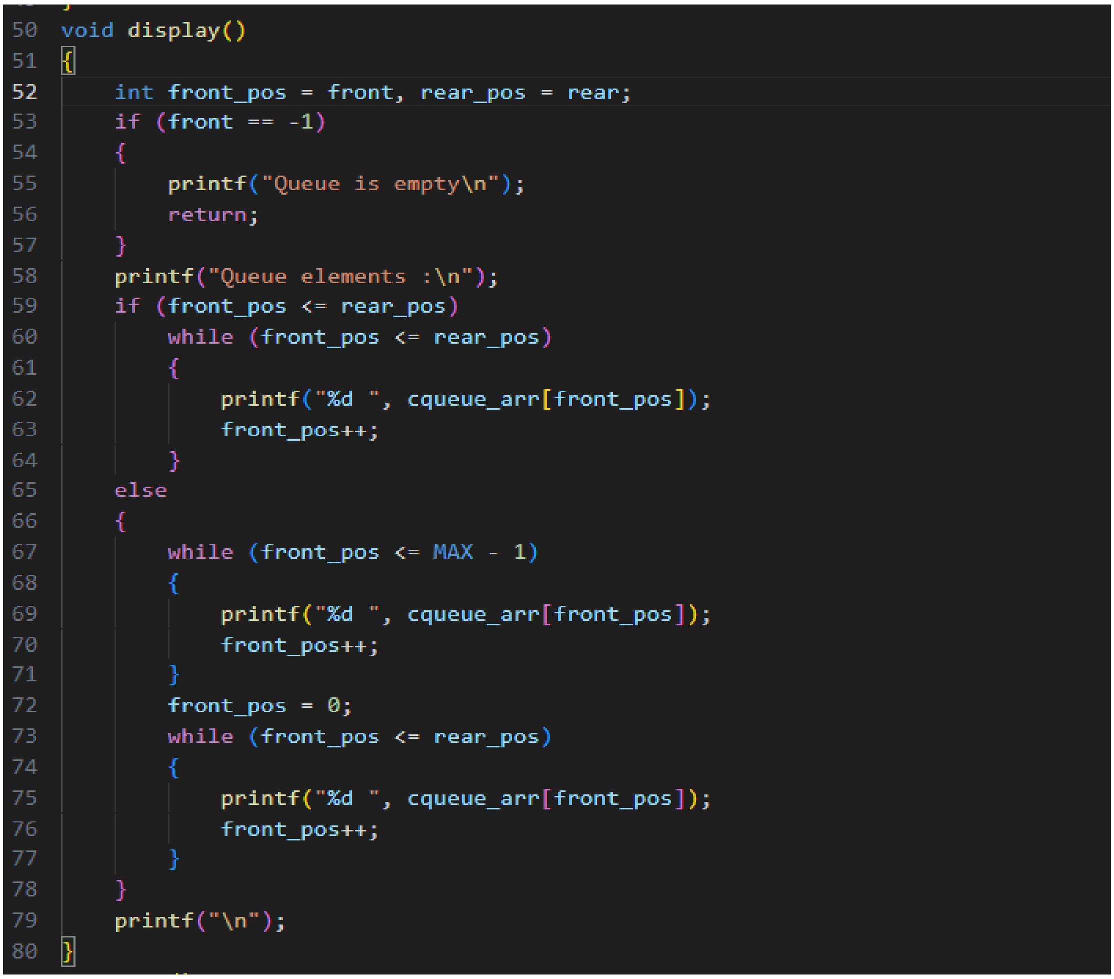
```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int cqueue_arr[MAX];
int front = -1;
int rear = -1;
void insert(int item)
{
    if ((front == 0 && rear == MAX - 1) || (front == rear + 1))
    {
        printf("Queue Overflow \n");
        return;
    }
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        if (rear == MAX - 1)
            rear = 0;
        else
            rear = rear + 1;
    }
    cqueue_arr[rear] = item;
    printf("item inserted in a queue is : %d\n", item);
}
```

Deletion function



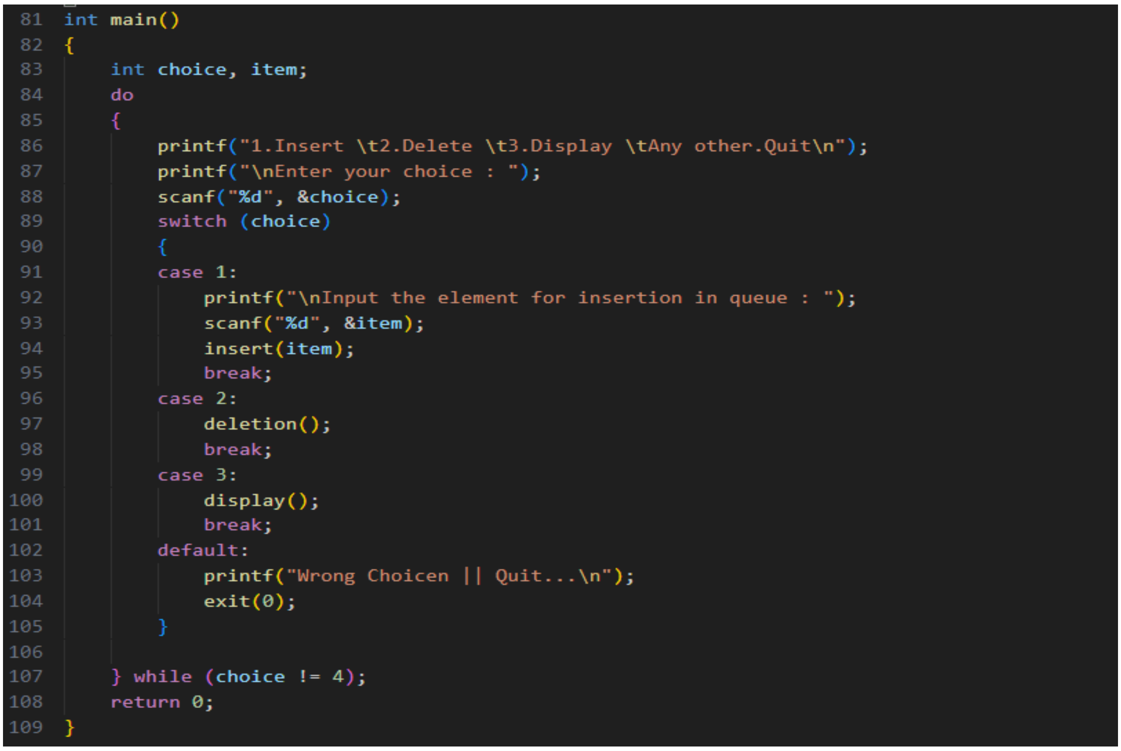
```c
void deletion()
{
    if (front == -1)
    {
        printf("Queue Underflown\n");
        return;
    }
    printf("Element deleted from queue is : %d\n", cqueue_arr[front]);
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else
    {
        if (front == MAX - 1)
            front = 0;
        else
            front = front + 1;
    }
}
```

Display Function



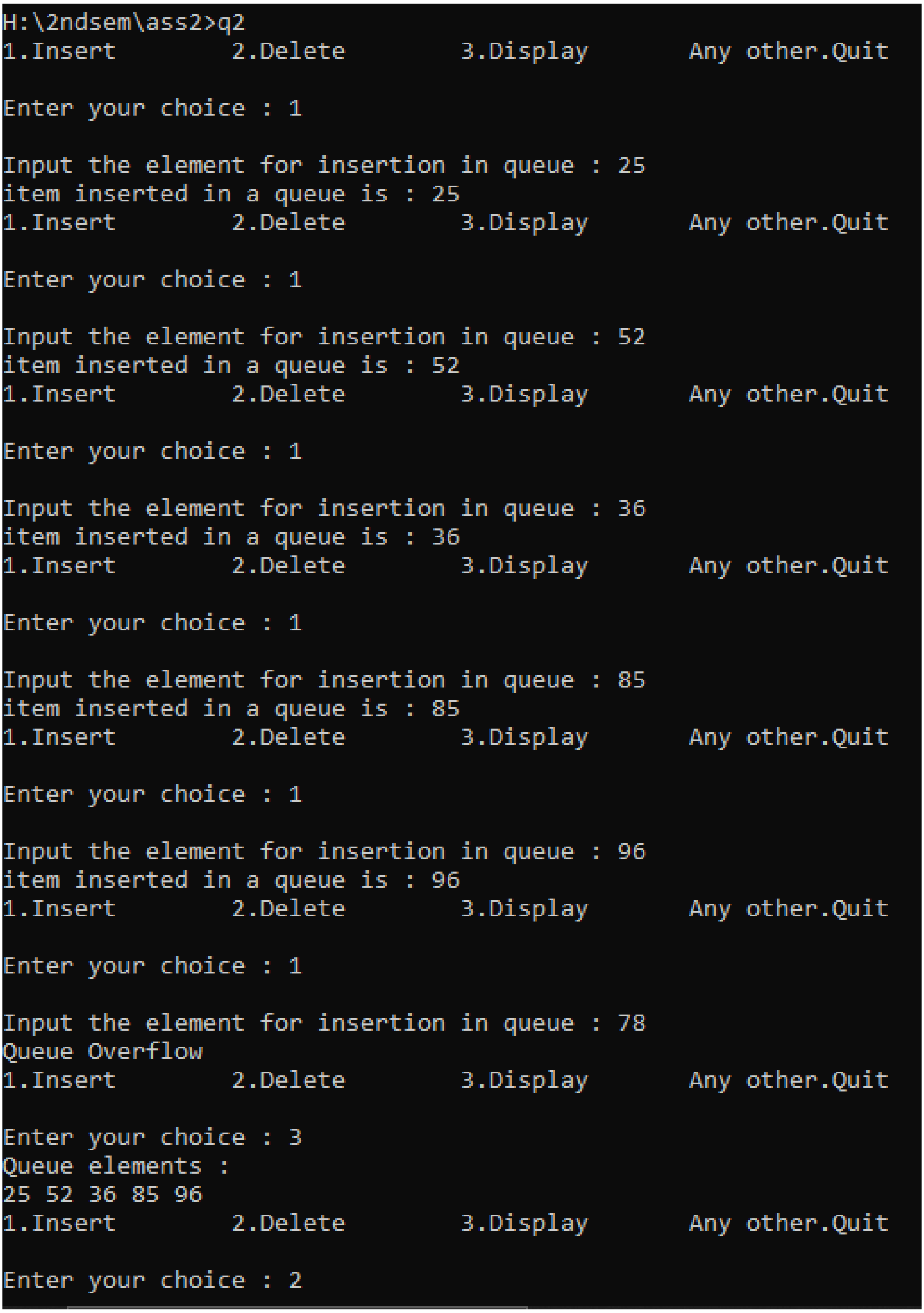
```c
void display()
{
    int front_pos = front, rear_pos = rear;
    if (front == -1)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n");
    if (front_pos <= rear_pos)
        while (front_pos <= rear_pos)
        {
            printf("%d ", cqueue_arr[front_pos]);
            front_pos++;
        }
    else
    {
        while (front_pos <= MAX - 1)
        {
            printf("%d ", cqueue_arr[front_pos]);
            front_pos++;
        }
        front_pos = 0;
        while (front_pos <= rear_pos)
        {
            printf("%d ", cqueue_arr[front_pos]);
            front_pos++;
        }
    }
    printf("\n");
}
```

Main Function

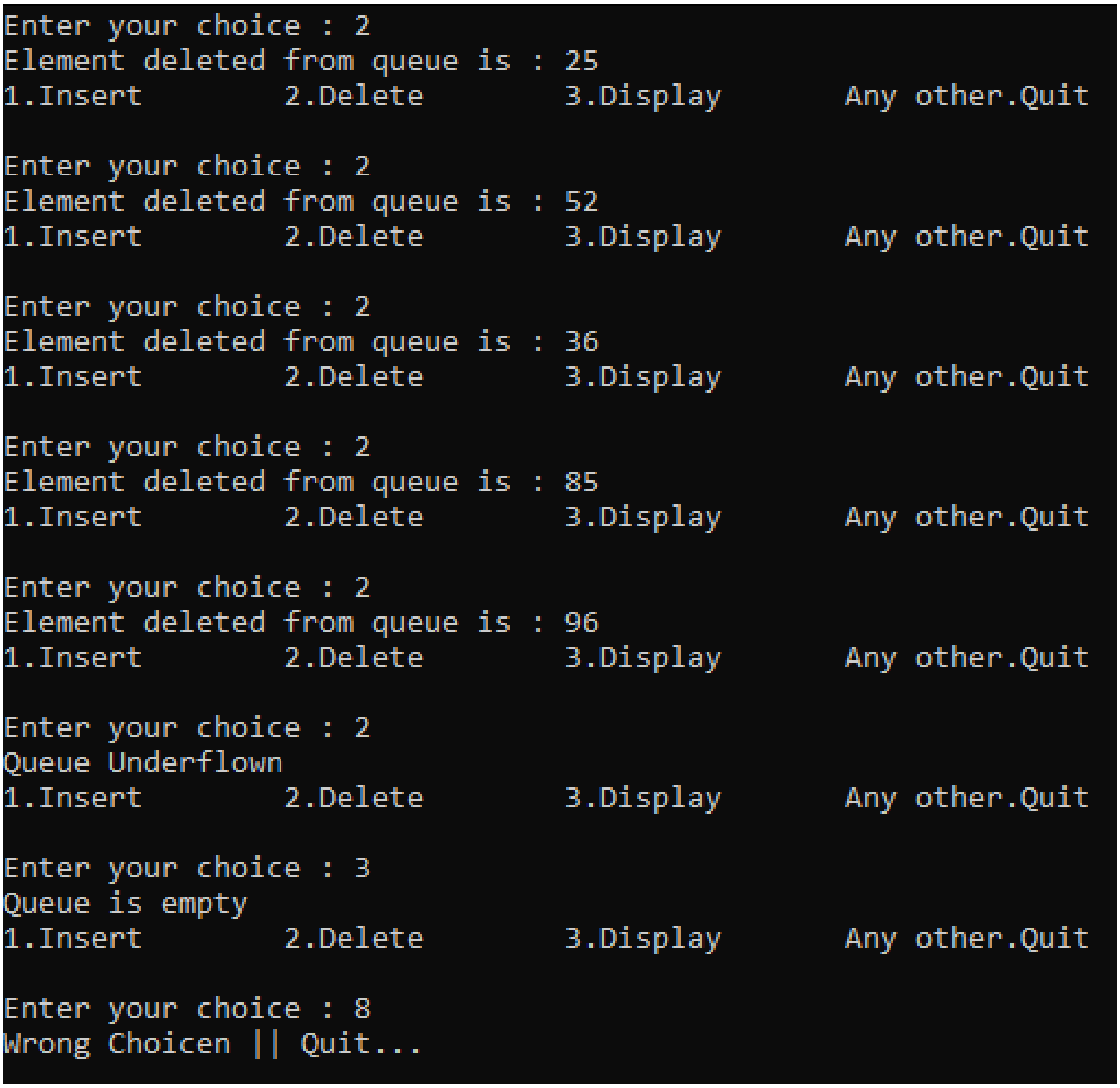


```c
int main()
{
    int choice, item;
    do
    {
        printf("1.Insert \t2.Delete \t3.Display \tAny other.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nInput the element for insertion in queue : ");
            scanf("%d", &item);
            insert(item);
            break;
        case 2:
            deletion();
            break;
        case 3:
            display();
            break;
        default:
            printf("Wrong Choicen || Quit...\n");
            exit(0);
        }

    } while (choice != 4);
    return 0;
}
```

Output



```
H:\2ndsem\ass2>q2
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 25
item inserted in a queue is : 25
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 52
item inserted in a queue is : 52
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 36
item inserted in a queue is : 36
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 85
item inserted in a queue is : 85
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 96
item inserted in a queue is : 96
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 1

Input the element for insertion in queue : 78
Queue Overflow
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 3
Queue elements :
25 52 36 85 96
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 2
```



```
Enter your choice : 2
Element deleted from queue is : 25
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 2
Element deleted from queue is : 52
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 2
Element deleted from queue is : 36
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 2
Element deleted from queue is : 85
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 2
Element deleted from queue is : 96
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 2
Queue Underflown
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 3
Queue is empty
1.Insert        2.Delete        3.Display       Any other.Quit

Enter your choice : 8
Wrong Choicen || Quit...
```

3. Write a program to perform following operations on Link List

a) Insertion

at the beginning

at the end

at the given location

in the sorted list

b) Deletion

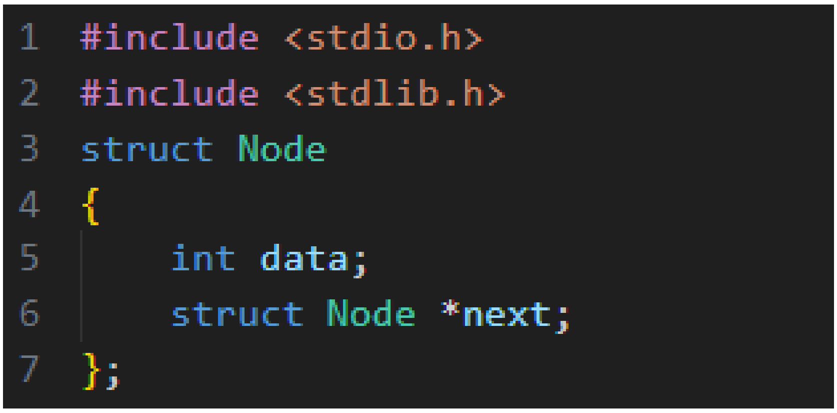
of first node

of last node

of given item of node

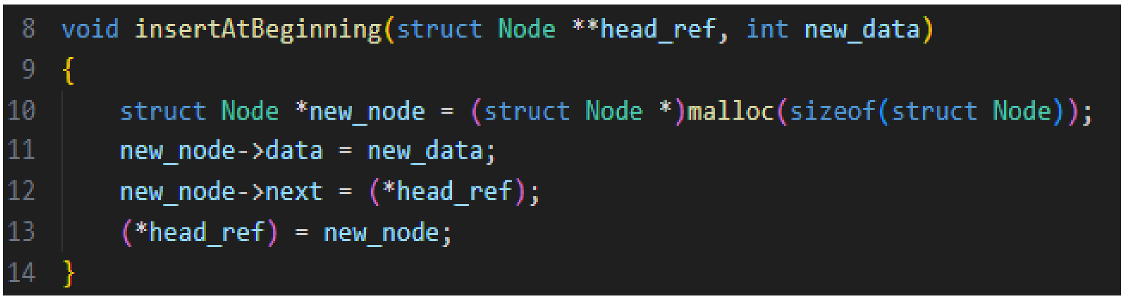
of given item from sorted list

Declaration section



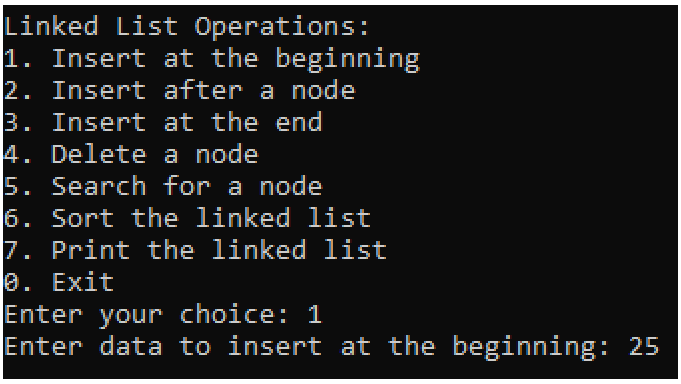
```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
};
```

Insert at beginning function



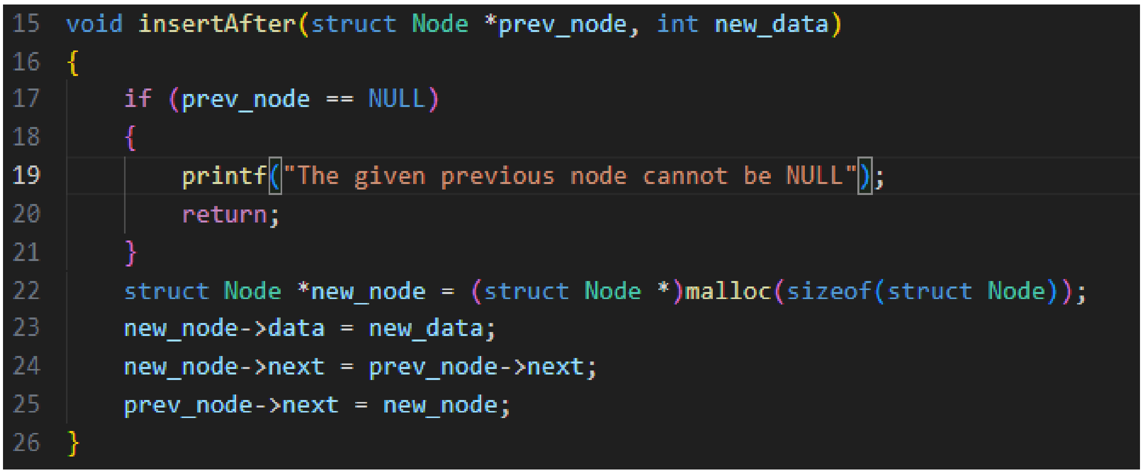
```c
void insertAtBeginning(struct Node **head_ref, int new_data)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
```

Output



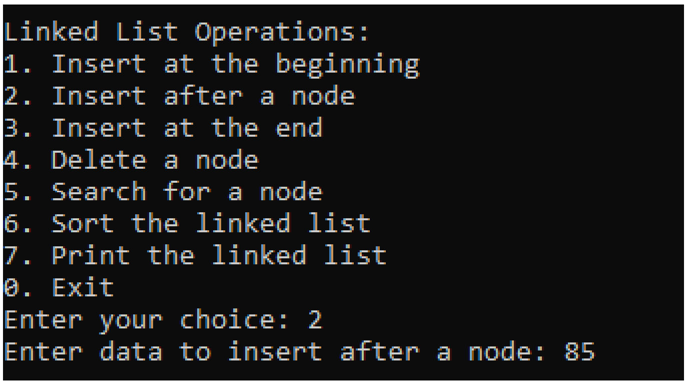
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 1
Enter data to insert at the beginning: 25
```

Insert after Function



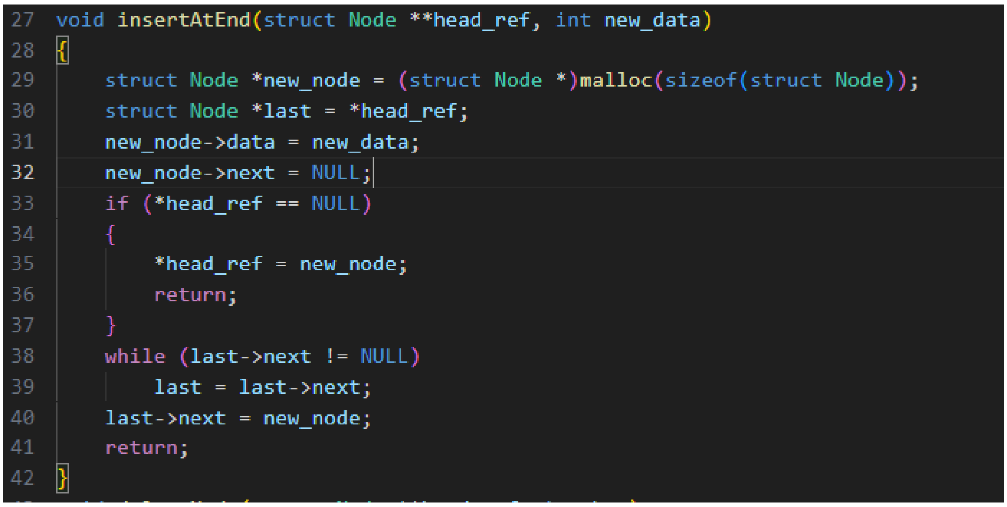
```c
void insertAfter(struct Node *prev_node, int new_data)
{
    if (prev_node == NULL)
    {
        printf("The given previous node cannot be NULL");
        return;
    }
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}
```

Output



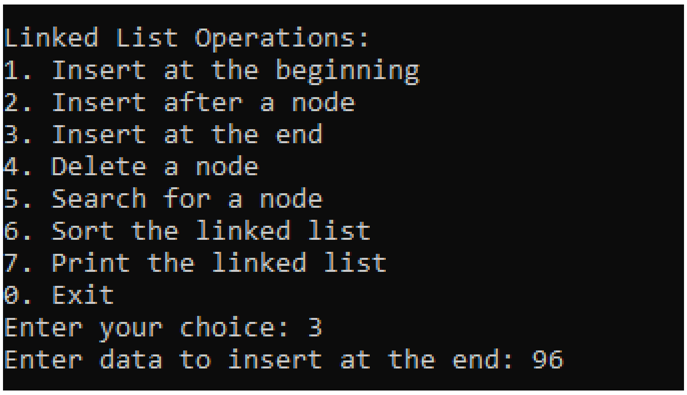
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 2
Enter data to insert after a node: 85
```

Insert at end



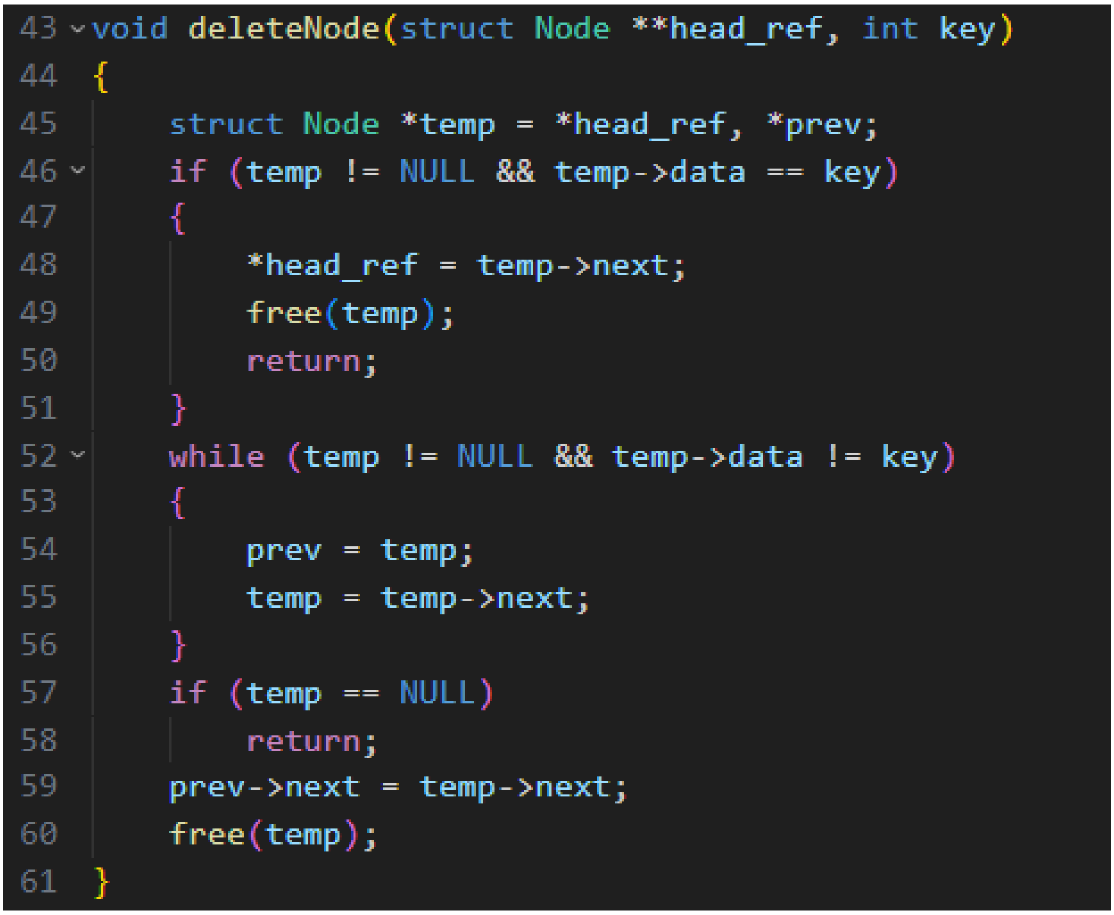
```c
void insertAtEnd(struct Node **head_ref, int new_data)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    struct Node *last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
    return;
}
```

Output



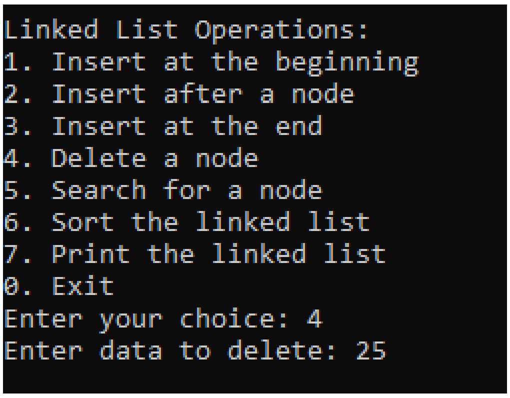
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 3
Enter data to insert at the end: 96
```

Delete node function



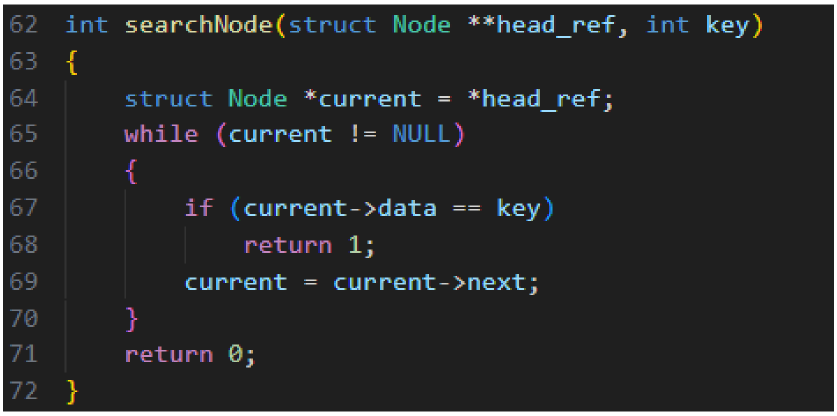
```c
void deleteNode(struct Node **head_ref, int key)
{
    struct Node *temp = *head_ref, *prev;
    if (temp != NULL && temp->data == key)
    {
        *head_ref = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL)
        return;
    prev->next = temp->next;
    free(temp);
}
```

Output



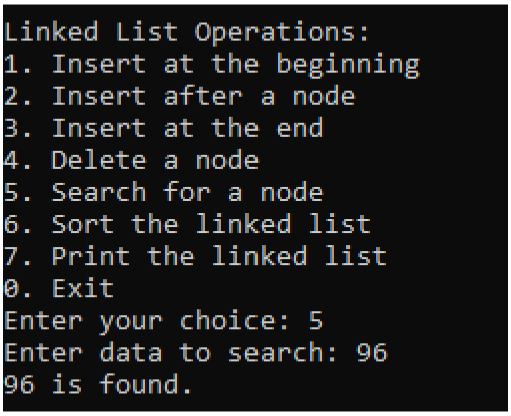
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 4
Enter data to delete: 25
```

Search node function



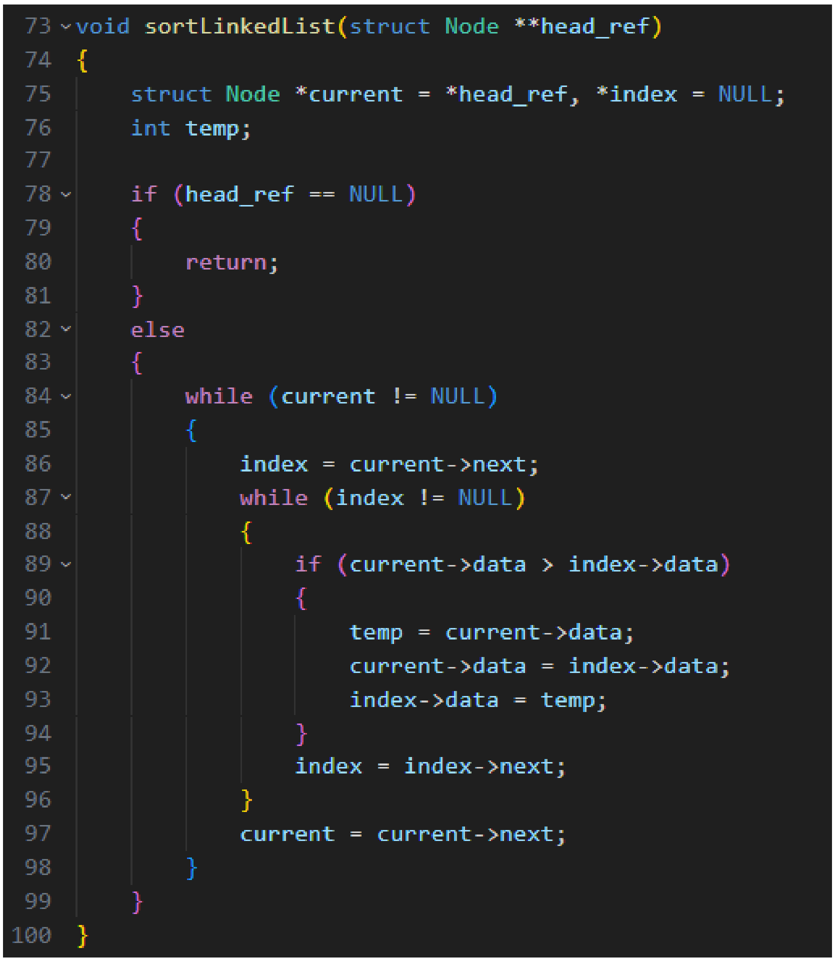
```c
int searchNode(struct Node **head_ref, int key)
{
    struct Node *current = *head_ref;
    while (current != NULL)
    {
        if (current->data == key)
            return 1;
        current = current->next;
    }
    return 0;
}
```

Output:-



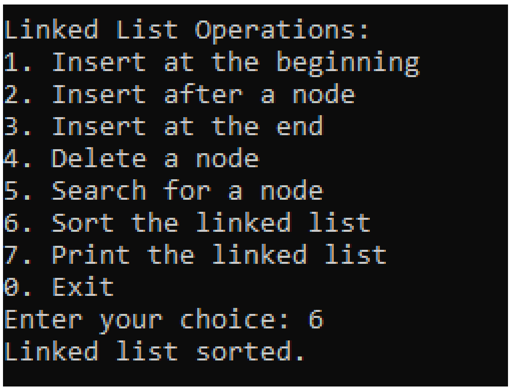
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 5
Enter data to search: 96
96 is found.
```

Sort linked list function



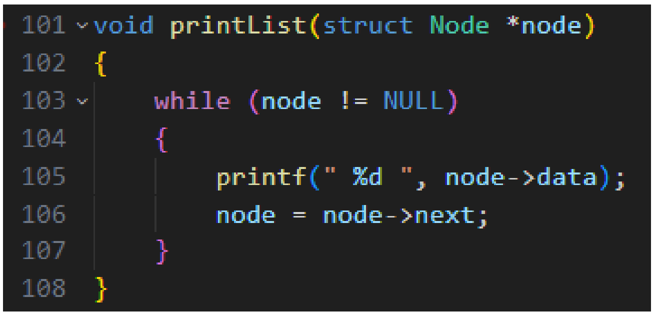
```c
void sortLinkedList(struct Node **head_ref)
{
    struct Node *current = *head_ref, *index = NULL;
    int temp;

    if (head_ref == NULL)
    {
        return;
    }
    else
    {
        while (current != NULL)
        {
            index = current->next;
            while (index != NULL)
            {
                if (current->data > index->data)
                {
                    temp = current->data;
                    current->data = index->data;
                    index->data = temp;
                }
                index = index->next;
            }
            current = current->next;
        }
    }
}
```

Output:-



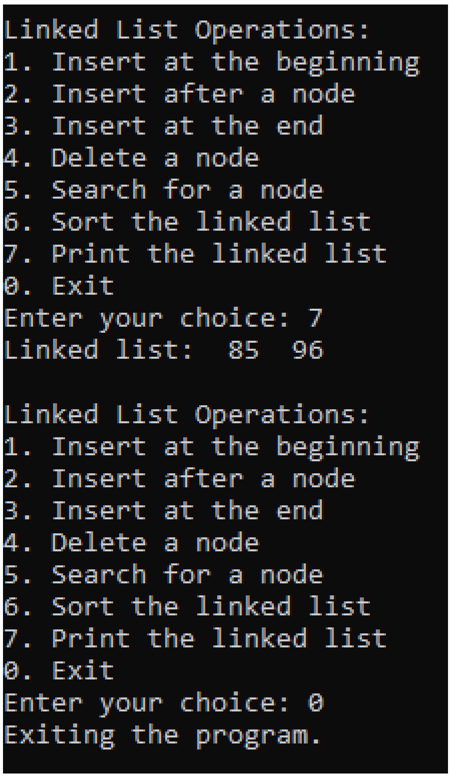
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 6
Linked list sorted.
```

Print linked list function



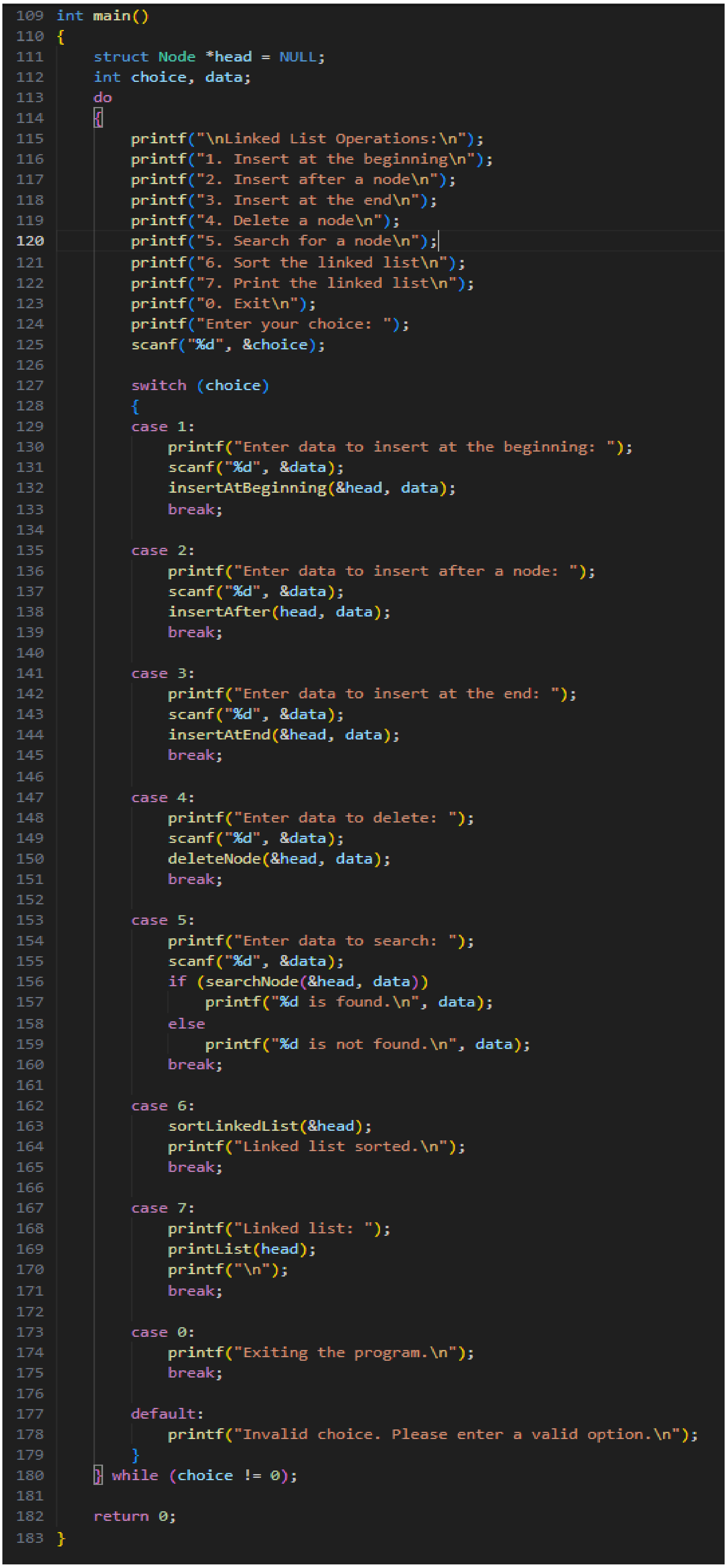
```c
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf(" %d ", node->data);
        node = node->next;
    }
}
```

Output:-



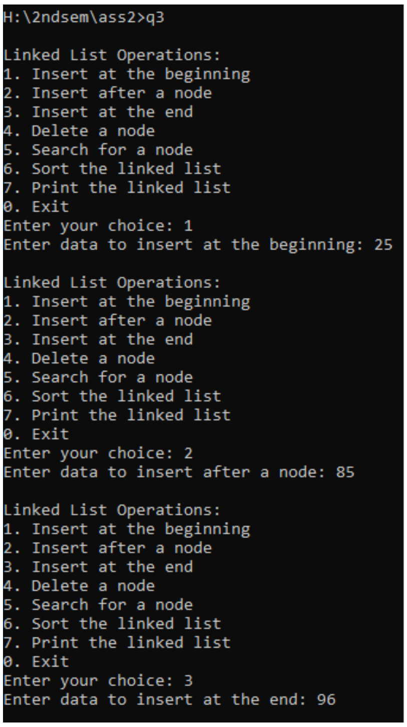
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 7
Linked list:  85  96

Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 0
Exiting the program.
```

Main function

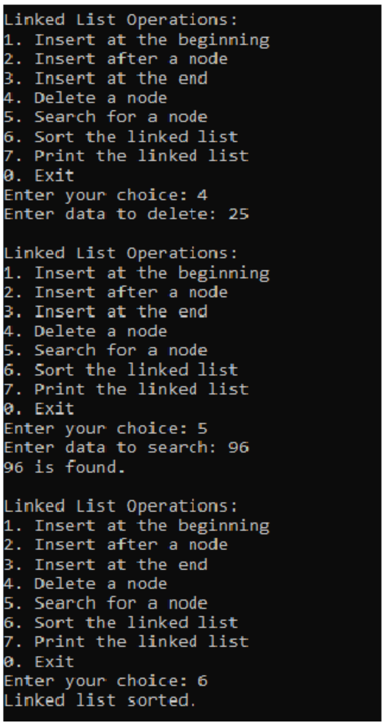


```c
int main()
{
    struct Node *head = NULL;
    int choice, data;
    do
    {
        printf("\nLinked List Operations:\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert after a node\n");
        printf("3. Insert at the end\n");
        printf("4. Delete a node\n");
        printf("5. Search for a node\n");
        printf("6. Sort the linked list\n");
        printf("7. Print the linked list\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter data to insert at the beginning: ");
            scanf("%d", &data);
            insertAtBeginning(&head, data);
            break;

        case 2:
            printf("Enter data to insert after a node: ");
            scanf("%d", &data);
            insertAfter(head, data);
            break;

        case 3:
            printf("Enter data to insert at the end: ");
            scanf("%d", &data);
            insertAtEnd(&head, data);
            break;

        case 4:
            printf("Enter data to delete: ");
            scanf("%d", &data);
            deleteNode(&head, data);
            break;

        case 5:
            printf("Enter data to search: ");
            scanf("%d", &data);
            if (searchNode(&head, data))
                printf("%d is found.\n", data);
            else
                printf("%d is not found.\n", data);
            break;

        case 6:
            sortLinkedList(&head);
            printf("Linked list sorted.\n");
            break;

        case 7:
            printf("Linked list: ");
            printList(head);
            printf("\n");
            break;

        case 0:
            printf("Exiting the program.\n");
            break;

        default:
            printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 0);

    return 0;
}
```

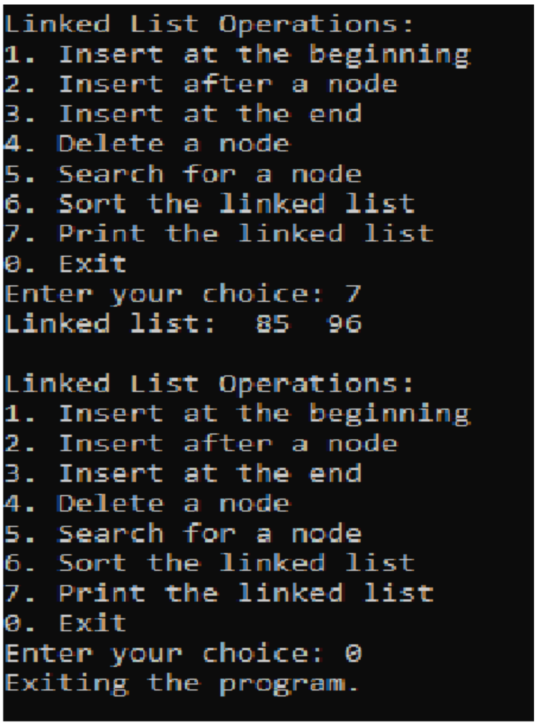
Output



```
H:\2ndsem\ass2>q3

Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 1
Enter data to insert at the beginning: 25

Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 2
Enter data to insert after a node: 85

Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 3
Enter data to insert at the end: 96
```

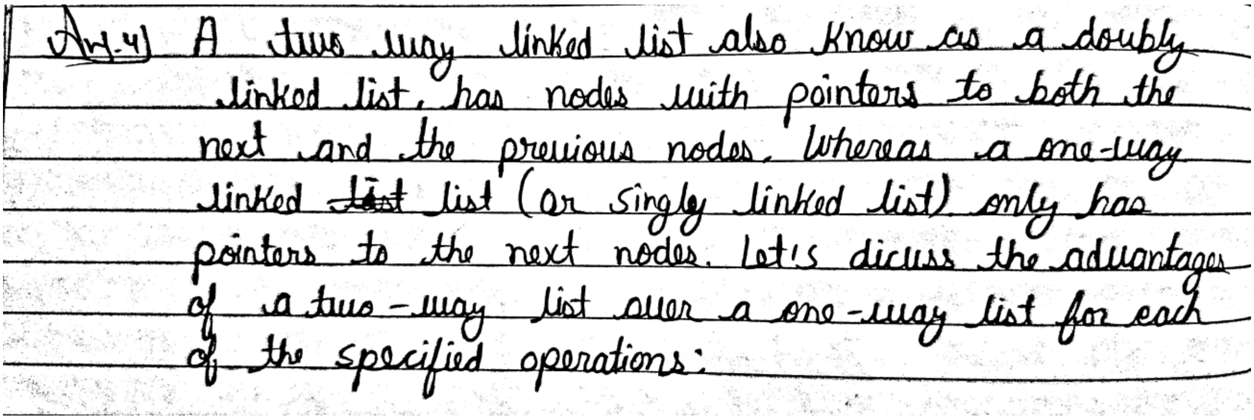


```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 4
Enter data to delete: 25

Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 5
Enter data to search: 96
96 is found.

Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 6
Linked list sorted.
```



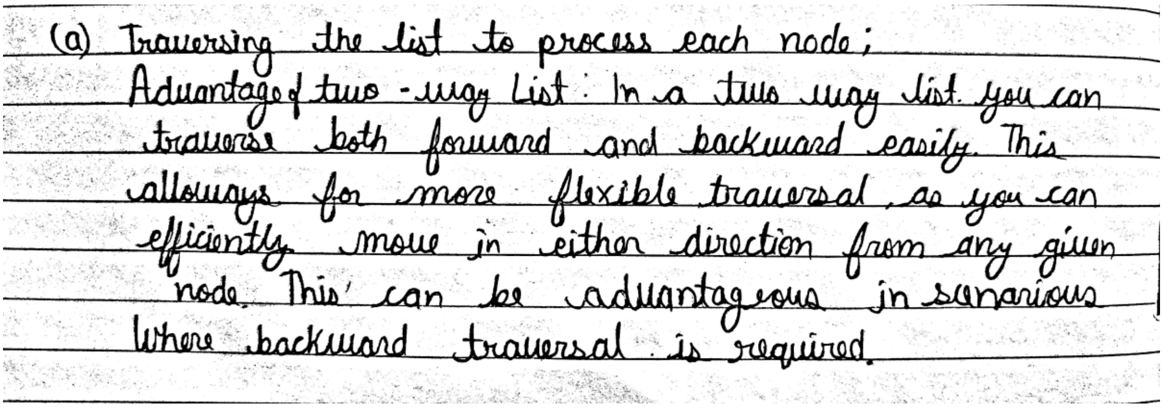
```
Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 7
Linked list:  85  96

Linked List Operations:
1. Insert at the beginning
2. Insert after a node
3. Insert at the end
4. Delete a node
5. Search for a node
6. Sort the linked list
7. Print the linked list
0. Exit
Enter your choice: 0
Exiting the program.
```

4. Discuss the advantages, if any, of a two-way list over a one way list for each of the following operations:



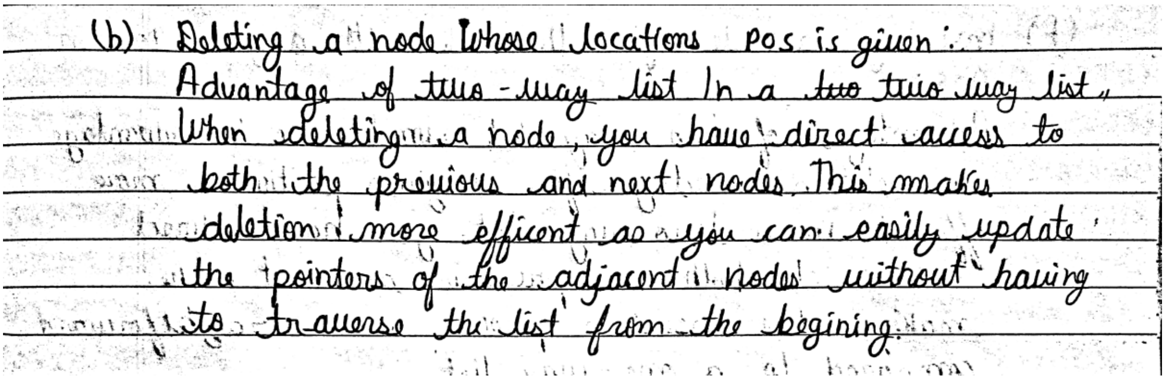
Ans-4) A two way linked list also Know as a doubly linked list, has nodes with pointers to both the next and the previous nodes. Whereas a one-way linked ~~list~~ list (or singly linked list) only has pointers to the next nodes. Let's dicuss the advantages of a two-way list over a one-way list for each of the specified operations:

a. Traversing the list to process each node.



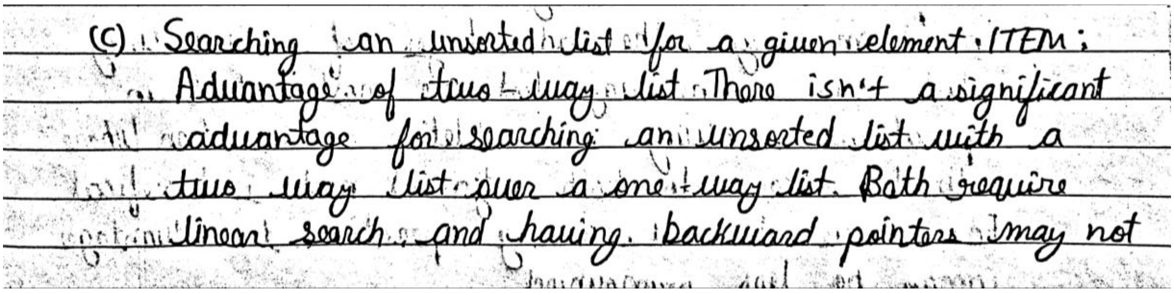
(a) Traversing the list to process each node;
Advantage of two-way List: In a two way list you can traverse both forward and backward easily. This allways for more flexible traversal, as you can efficiently move in either direction from any given node. This can be advantageous in scenarious Where backward traversal is required.

b. Deleting a node whose location POS(position of node) is given.



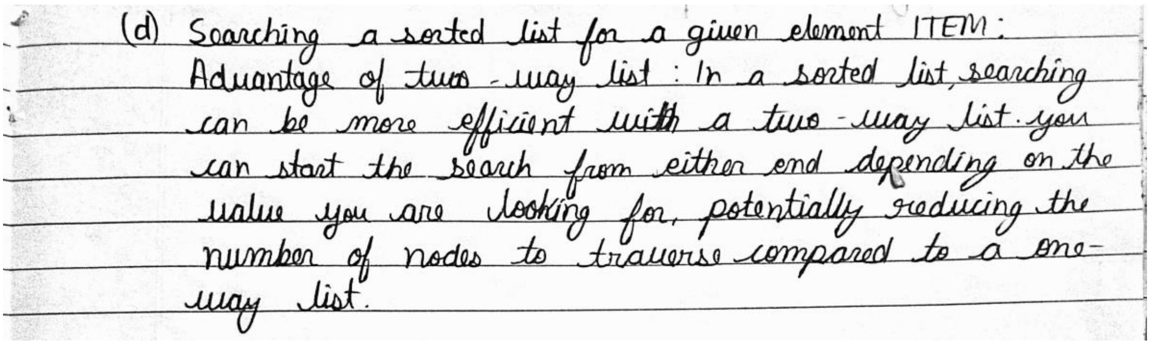
(b) Deleting a node Whose Locations pos is given:
Advantage of two-way list In a ~~two~~ two way list, When deleting a node, you have direct access to both the previous and next nodes. This makes deletion more efficent as you can easily update the pointers of the adjacent nodes without having to traverse the list from the begining.

c. Searching an unsorted list for a given elements ITEM.



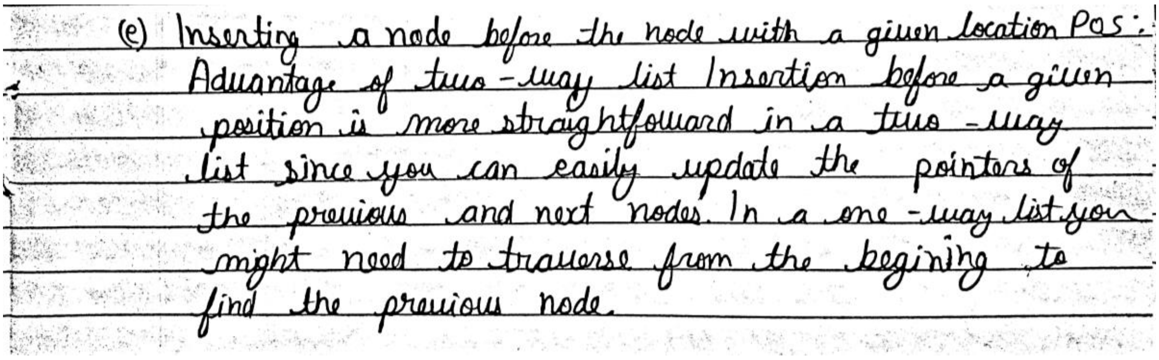
(C) Searching an unsorted list for a given element ITEM;
Advantage of two-way list There isn't a significant advantage for searching an unsorted list with a two way list over a one-way list. Both require linear search and having backward pointers may not

d. Searching a sorted list for a given elements ITEM.



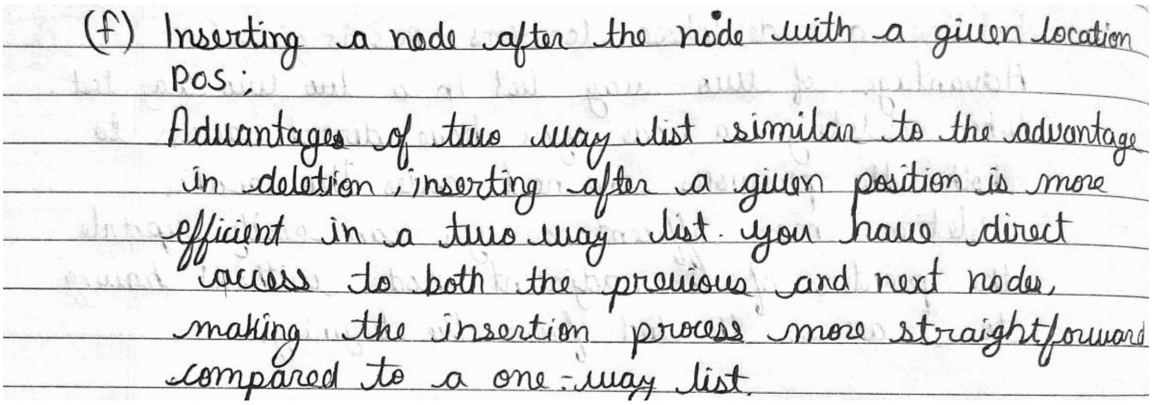
(d) Searching a sorted list for a given element ITEM:
Advantage of two-way list : In a sorted list, searching can be more efficient with a two-way list. you can start the search from either end depending on the value you are looking for, potentially reducing the number of nodes to traverse compared to a one-way list.

e. Inserting a node before the node with a given location POS



(e) Inserting a node before the node with a given location POS:
Advantage of two-way list Insertion before a given position is more straightfoward in a two-way list since you can easily update the pointers of the previous and next nodes. In a one-way list, you might need to traverse from the begining to find the previous node.

f. Inserting a node after the node with a given location POS



(f) Inserting a node after the node with a given location POS:
Advantages of two way list similar to the advantage in deletion, inserting after a given position is more efficient in a two way list. you have direct access to both the previous and next nodes, making the insertion process more straightforward compared to a one-way list.

Thankyou