Aim :- To Implement Linked list in c with basic operations like Add, Delete, Update, Search, Display.

Name :- SUJAL NIMJE

Roll no :- 64

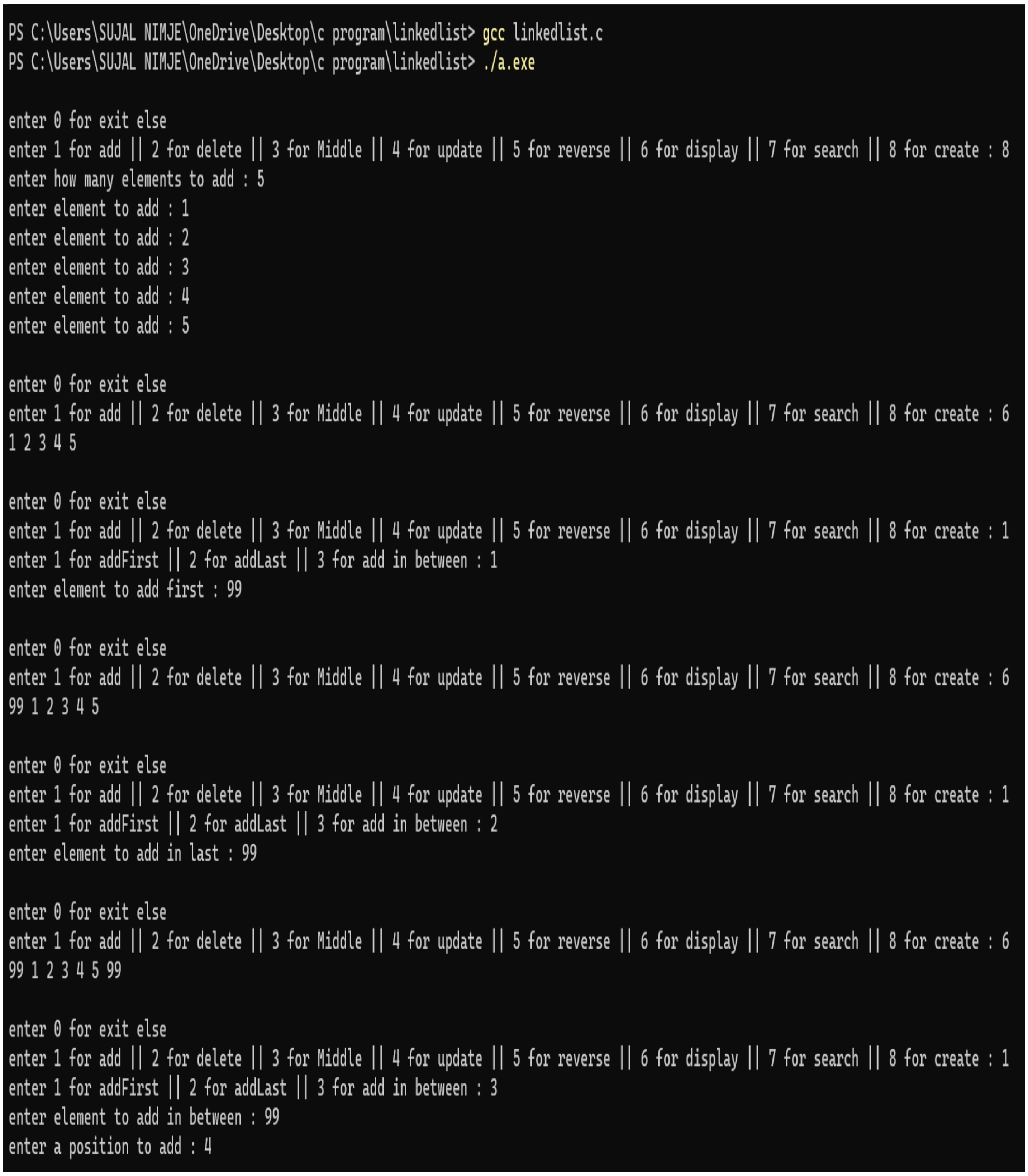
Subject :- DS

Code :-

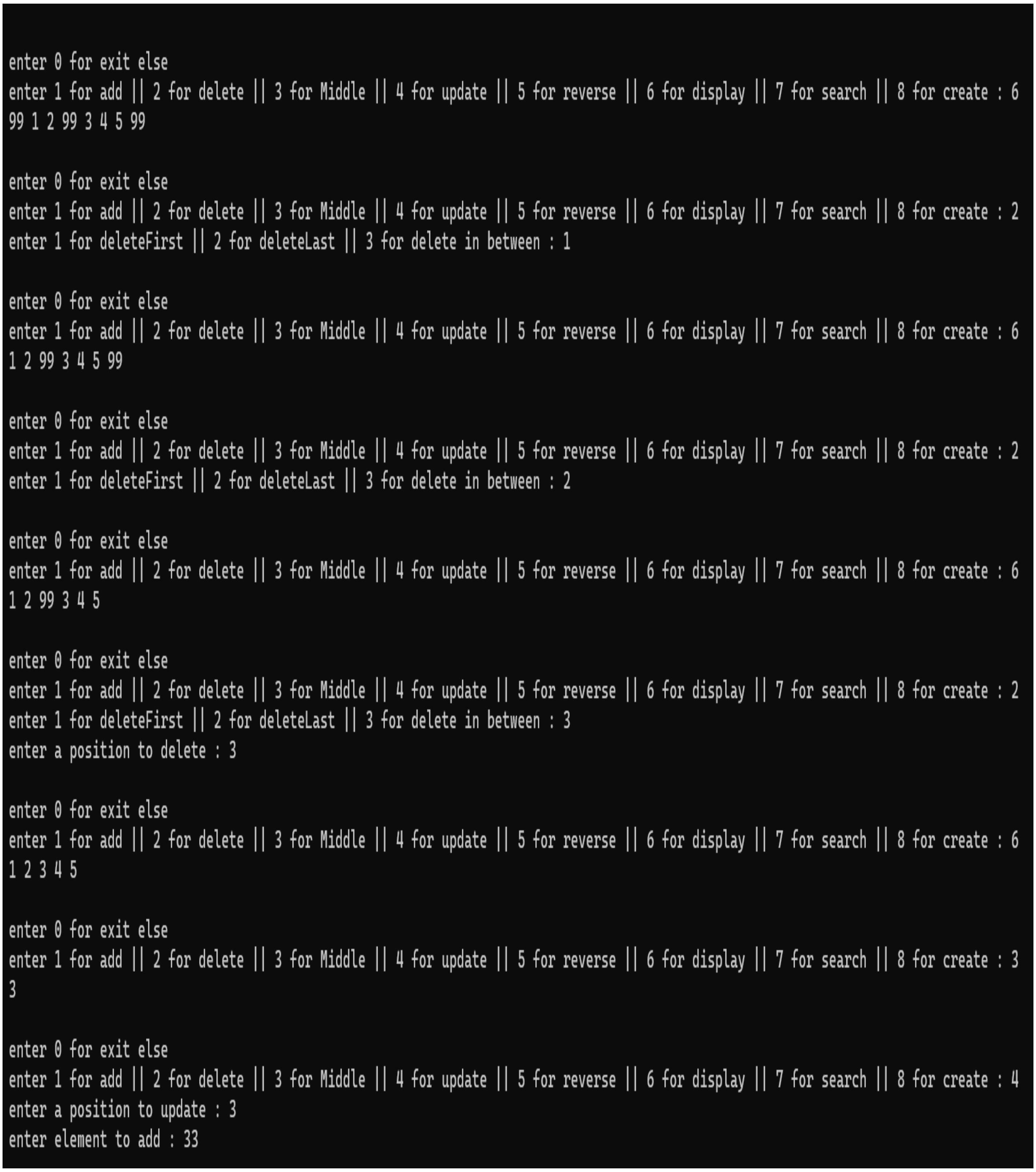
```c
#include <stdio.h>
#include <stdlib.h>

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

struct node *addLast(struct node *head, int data)
{

    struct node *temp = (struct node *)malloc(sizeof(struct node));

    temp->data = data;
    temp->next = NULL;

    if (head == NULL)
    {
        return temp;
    }
    struct node *ptr = head;
    while (ptr->next != NULL)
    {
        ptr = ptr->next;
    }
    ptr->next = temp;
    return head;
}
struct node *addFirst(struct node *head, int element)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = element;
    temp->next = NULL;
    temp->next = head;
    return temp;
}
struct node *add(struct node *head, int element, int position)
```

```c
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = element;
    temp->next = NULL;

    if (head == NULL)
    {
        return temp;
    }
    int i = 1;
    struct node *prev = head;
    struct node *second = head->next;
    while (prev != NULL)
    {
        if (i == position - 1)
        {
            temp->next = second;
            prev->next = temp;
            return head;
        }
        i++;
        prev = prev->next;
        second = second->next;
    }
    return head;
}
void display(struct node *head)
{
    if (head == NULL)
    {
        printf("linked list is empty\n");
        return;
    }
    struct node *ptr = head;
    while (ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}

void findMiddle(struct node *head)
{
    if (head == NULL)
    {
        printf("linked list empty\n");
        return;
    }
    struct node *ptr1 = head;
    struct node *ptr2 = head;
    while (ptr2->next != NULL && ptr2->next->next != NULL)
    {
        ptr1 = ptr1->next;
        ptr2 = ptr2->next->next;
    }
    printf("%d\n", ptr1->data);
}

struct node *reverse(struct node *head)
{
    if (head == NULL || head->next == NULL)
    {
        return head;
    }
    struct node *newnode = reverse(head->next);
    head->next->next = head;
    head->next = NULL;
    return newnode;
}

struct node *deleteFirst(struct node *head)
{
    if (head == NULL)
    {
        printf("linked list is empty\n");
        return head;
    }
    return head->next;
}
struct node *deleteLast(struct node *head)
{
    if (head == NULL)
    {
        printf("linked list is empty\n");
        return NULL;
    }
    if (head->next == NULL)
    {
        return NULL;
    }
    struct node *curr = head;
    struct node *prev;
    while (curr->next != NULL)
    {
        prev = curr;
        curr = curr->next;
    }
    struct node *temp = curr;
    int num = curr->data;
    prev->next = NULL;
    free(temp);
    return head;
}

struct node *delete(struct node *head, int post)
{
    if (head == NULL)
    {
        printf("Linked List Empty\n");
        return head;
    }

    if (head->next == NULL)
    {
        return head->next;
    }

    struct node *curr = head->next;
    struct node *prev = head;
    while (post > 2)
    {
        if ((curr == NULL))
        {
            printf("index out of bound\n");
            return NULL;
        }
        prev = curr;
        curr = curr->next;
        post--;
    }
    struct node *temp = curr;
    int num = temp->data;
    prev->next = curr->next;
    free(temp);
    return head;
}

void update(struct node *head, int position, int element)
{
    if (head == NULL)
    {
        printf("linked list is empty!\n");
        return;
    }
    struct node *curr = head;
    while (position > 1)
    {
        if (curr == NULL)
        {
            printf("index out of bound!\n");
            return;
        }
        position--;
        curr = curr->next;
    }
    curr->data = element;
}
void deallocateMemory(struct node *head)
{
    if (head == NULL)
    {
        return;
    }
    deallocateMemory(head->next);
    free(head);
}
int search(struct node *head, int element)
{
    if (head == NULL)
    {
        printf("linked list is empty\n");
        return -1;
    }

    struct node *curr = head;
    int count = 1;
    while (curr != NULL)
    {
        if (element == curr->data)
        {
            return count;
        }
        curr = curr->next;
        count++;
    }
    return -1;
}
void main()
{

    struct node *head = NULL;
    int choice;
    do
    {
        printf("\nenter 0 for exit else \nenter 1 for add || 2 for delete || 3 for Middle
|| 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            int x;
            printf("enter 1 for addFirst || 2 for addLast || 3 for add in between : ");
            scanf("%d", &x);
            switch (x)
            {
            case 1:
            {
                int num;
                printf("enter element to add first : ");
                scanf("%d", &num);
                head = addFirst(head, num);
                break;
            }
            case 2:
            {
                int num;
                printf("enter element to add in last : ");
                scanf("%d", &num);
                head = addLast(head, num);
                break;
            }
            case 3:
            {
                int post, num;
                printf("enter element to add in between : ");
                scanf("%d", &num);
                printf("enter a position to add : ");
                scanf("%d", &post);
                head = add(head, num, post);
                break;
            }
            }
            break;
        }
        case 2:
        {
            int x;
            printf("enter 1 for deleteFirst || 2 for deleteLast || 3 for delete in between
: ");

            scanf("%d", &x);

            switch (x)
            {
            case 1:
            {
                head = deleteFirst(head);
                break;
            }
            case 2:
            {
                head = deleteLast(head);
                break;
            }
            case 3:
            {
                int post;
                printf("enter a position to delete : ");
                scanf("%d", &post);
                head = delete (head, post);
                break;
            }
            }
            break;
        }
        case 3:
        {
            findMiddle(head);
            break;
        }
        case 4:
        {
            int post, x;
            printf("enter a position to update : ");
            scanf("%d", &post);
            printf("enter element to add : ");
            scanf("%d", &x);
            update(head, post, x);
            break;
        }
        case 5:
        {
            head = reverse(head);
            break;
        }
        case 6:
        {
            display(head);
            break;
        }
        case 7:
        {
            int x;
            printf("enter element to search : ");
            scanf("%d", &x);
            printf("%d is present in position %d", x, search(head, x));
            break;
        }
        case 8:
        {
            int size, num;
            printf("enter how many elements to add : ");
            scanf("%d", &size);
            for (int i = 0; i < size; i++)
            {
                printf("enter element to add : ");
                scanf("%d", &num);
                head = addLast(head, num);
            }
            break;
        }
        case 0:
        {
            deallocateMemory(head);
        }
        default:
            printf("an invalid input\n");
        }
    } while (choice != 0);
}
```

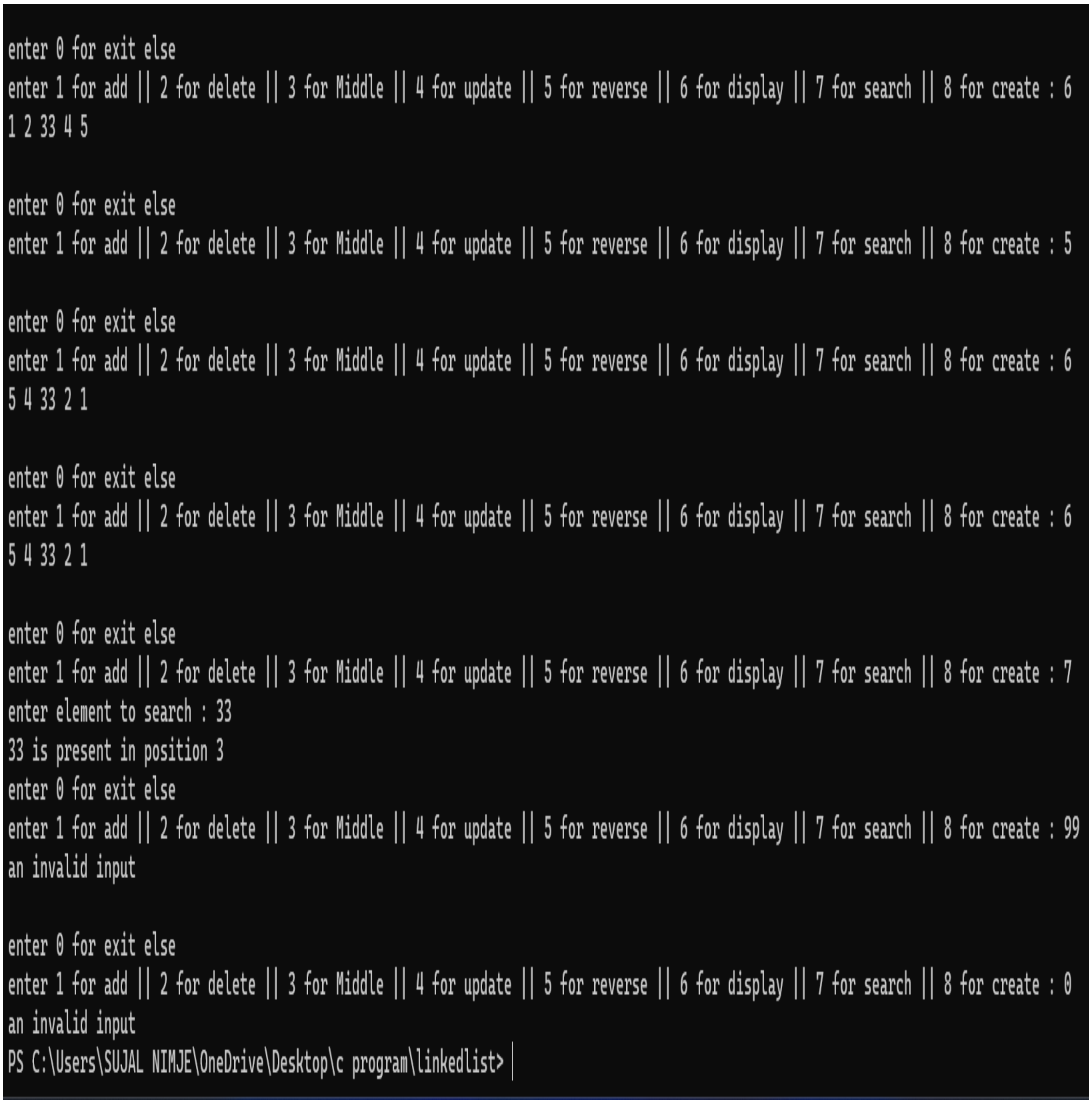
Output :-



```
PS C:\Users\SUJAL NIMJE\OneDrive\Desktop\c program\linkedlist> gcc linkedlist.c
PS C:\Users\SUJAL NIMJE\OneDrive\Desktop\c program\linkedlist> ./a.exe

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 8
enter how many elements to add : 5
enter element to add : 1
enter element to add : 2
enter element to add : 3
enter element to add : 4
enter element to add : 5

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
1 2 3 4 5

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 1
enter 1 for addFirst || 2 for addLast || 3 for add in between : 1
enter element to add first : 99

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
99 1 2 3 4 5

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 1
enter 1 for addFirst || 2 for addLast || 3 for add in between : 2
enter element to add in last : 99

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
99 1 2 3 4 5 99

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 1
enter 1 for addFirst || 2 for addLast || 3 for add in between : 3
enter element to add in between : 99
enter a position to add : 4
```



```
enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
99 1 2 99 3 4 5 99

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 2
enter 1 for deleteFirst || 2 for deleteLast || 3 for delete in between : 1

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
1 2 99 3 4 5 99

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 2
enter 1 for deleteFirst || 2 for deleteLast || 3 for delete in between : 2

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
1 2 99 3 4 5

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 2
enter 1 for deleteFirst || 2 for deleteLast || 3 for delete in between : 3
enter a position to delete : 3

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
1 2 3 4 5

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 3
3

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 4
enter a position to update : 3
enter element to add : 33
```



```
enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
1 2 33 4 5

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 5

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
5 4 33 2 1

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 6
5 4 33 2 1

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 7
enter element to search : 33
33 is present in position 3
enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 99
an invalid input

enter 0 for exit else
enter 1 for add || 2 for delete || 3 for Middle || 4 for update || 5 for reverse || 6 for display || 7 for search || 8 for create : 0
an invalid input
PS C:\Users\SUJAL NIMJE\OneDrive\Desktop\c program\linkedlist>
```