**BCSE202P -**

**Data Structures and Algorithms**

Digital Assignment – 2

**Name:** Dhruv Rajeshkumar Shah

**Registration No –** 21BCE0611

1 .Write a menu driven program to perform following functions in a singly linked list.

i) Insertion in the beginning of the list

ii) Insertion at the end of the list

iii) Insertion in a particular location of the list

iv) Deletion based on a particular value

v) Deletion based on a particular location

vi) Deleting an element at the beginning

vii) Deleting an element at the end of the list

viii) Search an element

ix) Reverse the list

x) Count the number of even and odd numbers in the list

xi) Display the contents of the list

**CODE**

// DSA

// DA-2

// Dhruv Rajeshkumar Shah

// 21BCE0611

#include <stdio.h>

#include <stdlib.h>

// Linkedlist node

struct node

{

    int value;

    struct node \*next;

};

// Function to create a new node

struct node \*newNode(int value)

{

    struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

    new\_node->value = value;

    new\_node->next = NULL;

    return new\_node;

}

// Function to insert a new node at the beginning of the linkedlist

struct node \*insertAtBeginning(struct node \*head, int value)

{

    struct node \*new\_node = newNode(value);

    new\_node->next = head;

    head = new\_node;

    return head;

}

// Function to insert a new node at the end of the linkedlist

struct node \*insertAtEnd(struct node \*head, int value)

{

    struct node \*new\_node = newNode(value);

    if (head == NULL)

    {

        head = new\_node;

        return head;

    }

    struct node \*temp = head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    temp->next = new\_node;

    return head;

}

// Function to insert a new node at the given position of the linkedlist

struct node \*insertAtPosition(struct node \*head, int value, int position)

{

    struct node \*new\_node = newNode(value);

    if (position == 1)

    {

        new\_node->next = head;

        head = new\_node;

        return head;

    }

    struct node \*temp = head;

    for (int i = 1; i < position - 1; i++)

    {

        temp = temp->next;

    }

    new\_node->next = temp->next;

    temp->next = new\_node;

    return head;

}

// Function to delete a node for a given value

struct node \*deleteNode(struct node \*head, int value)

{

    struct node \*temp = head;

    if (temp->value == value)

    {

        head = temp->next;

        free(temp);

        return head;

    }

    while (temp->next->value != value)

    {

        temp = temp->next;

    }

    struct node \*temp2 = temp->next;

    temp->next = temp->next->next;

    free(temp2);

    return head;

}

// Function to delete a node for a given position

struct node \*deleteNodeAtPosition(struct node \*head, int position)

{

    struct node \*temp = head;

    if (position == 1)

    {

        head = temp->next;

        free(temp);

        return head;

    }

    for (int i = 1; i < position - 1; i++)

    {

        temp = temp->next;

    }

    struct node \*temp2 = temp->next;

    temp->next = temp->next->next;

    free(temp2);

    return head;

}

// Function to delete a node at the beginning of the linkedlist

struct node \*deleteAtBeginning(struct node \*head)

{

    struct node \*temp = head;

    head = temp->next;

    free(temp);

    return head;

}

// Function to delete a node at the end of the linkedlist

struct node \*deleteAtEnd(struct node \*head)

{

    struct node \*temp = head;

    while (temp->next->next != NULL)

    {

        temp = temp->next;

    }

    struct node \*temp2 = temp->next;

    temp->next = NULL;

    free(temp2);

    return head;

}

// Function to search for a node with a given value

int search(struct node \*head, int value)

{

    struct node \*temp = head;

    int position = 1;

    while (temp != NULL)

    {

        if (temp->value == value)

        {

            return position;

        }

        temp = temp->next;

        position++;

    }

    return -1;

}

// Function to reverse the linkedlist

struct node \*reverse(struct node \*head)

{

    struct node \*prev = NULL;

    struct node \*current = head;

    struct node \*next = NULL;

    while (current != NULL)

    {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    head = prev;

    return head;

}

// Function to count the number of odd and even numbers in the linkedlist

void countOddEven(struct node \*head)

{

    int odd = 0, even = 0;

    struct node \*temp = head;

    while (temp != NULL)

    {

        if (temp->value % 2 == 0)

        {

            even++;

        }

        else

        {

            odd++;

        }

        temp = temp->next;

    }

    printf("Number of odd numbers: %d\n", odd);

    printf("Number of even numbers: %d\n", even);

}

// Function to print the linkedlist

void printList(struct node \*head)

{

    struct node \*temp = head;

    printf("\nLinkedlist: ");

    while (temp != NULL)

    {

        printf("%d ", temp->value);

        temp = temp->next;

    }

    printf("\n\n");

}

// Menu driven program

int main()

{

    struct node \*head = NULL;

    int choice = 0;

    while (choice != 12)

    {

        printf("\nEnter a choice from the following:\n");

        printf("1. Insert at beginning\n");

        printf("2. Insert at end\n");

        printf("3. Insert at position\n");

        printf("4. Delete a node\n");

        printf("5. Delete a node at position\n");

        printf("6. Delete at beginning\n");

        printf("7. Delete at end\n");

        printf("8. Search for a node\n");

        printf("9. Reverse the linkedlist\n");

        printf("10. Count the number of odd and even numbers in the linkedlist\n");

        printf("11. Print the linkedlist\n");

        printf("12. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            printf("Enter the value to be inserted: ");

            int val;

            scanf("%d", &val);

            head = insertAtBeginning(head, val);

            break;

        case 2:

            printf("Enter the value to be inserted: ");

            int val2;

            scanf("%d", &val2);

            head = insertAtEnd(head, val2);

            break;

        case 3:

            printf("Enter the value to be inserted: ");

            int val3;

            scanf("%d", &val3);

            printf("Enter the position: ");

            int position3;

            scanf("%d", &position3);

            head = insertAtPosition(head, val3, position3);

            break;

        case 4:

            printf("Enter the value to be deleted: ");

            int val4;

            scanf("%d", &val4);

            head = deleteNode(head, val4);

            break;

        case 5:

            printf("Enter the position: ");

            int position5;

            scanf("%d", &position5);

            head = deleteNodeAtPosition(head, position5);

            break;

        case 6:

            head = deleteAtBeginning(head);

            break;

        case 7:

            head = deleteAtEnd(head);

            break;

        case 8:

            printf("Enter the value to be searched: ");

            int val8;

            scanf("%d", &val8);

            int position8 = search(head, val8);

            if (position8 == -1)

            {

                printf("Value not found\n");

            }

            else

            {

                printf("Value found at position %d\n", position8);

            }

            break;

        case 9:

            head = reverse(head);

            break;

        case 10:

            countOddEven(head);

            break;

        case 11:

            printList(head);

            break;

        case 12:

            break;

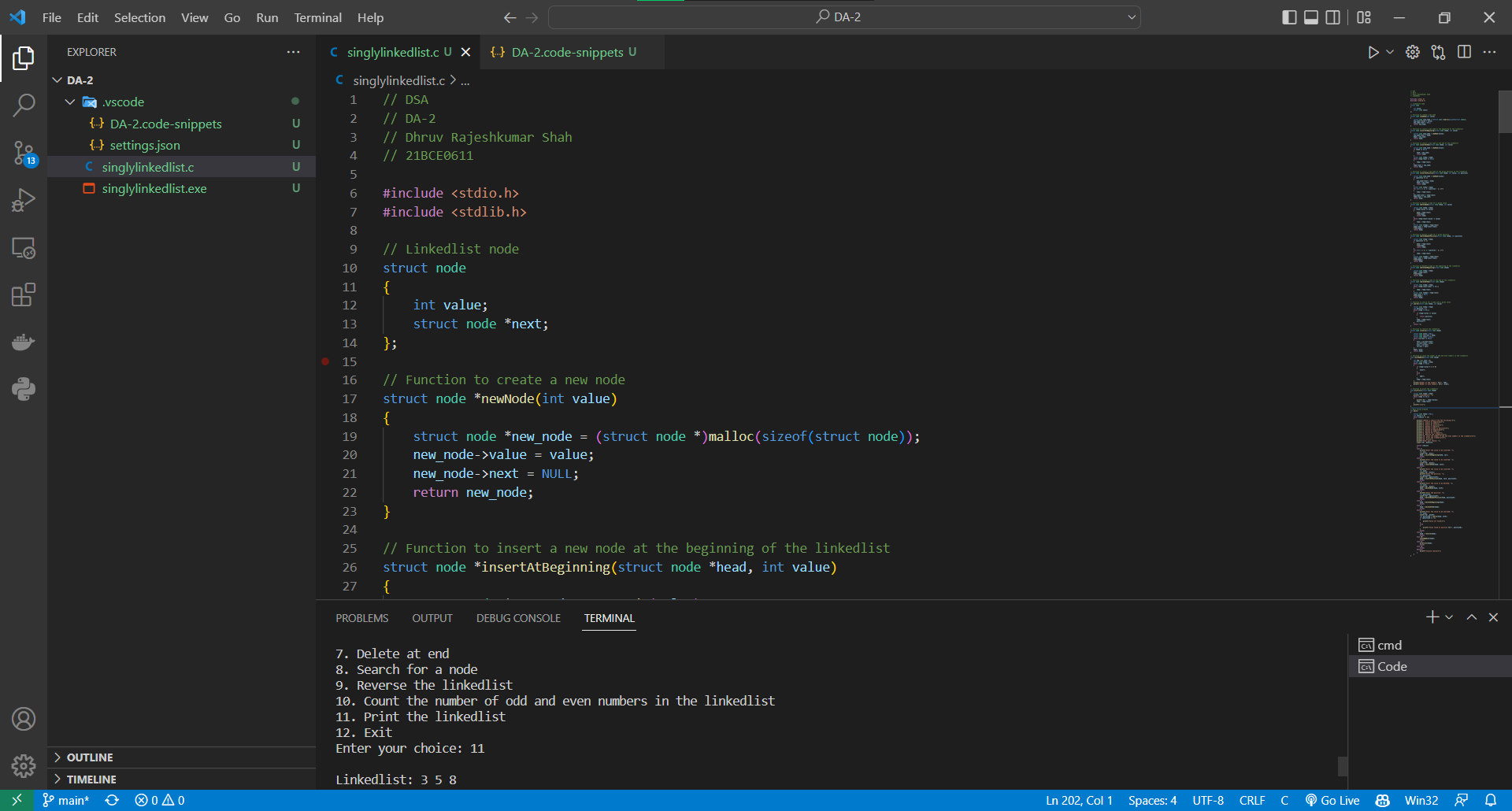
        default:

            printf("Invalid choice\n");

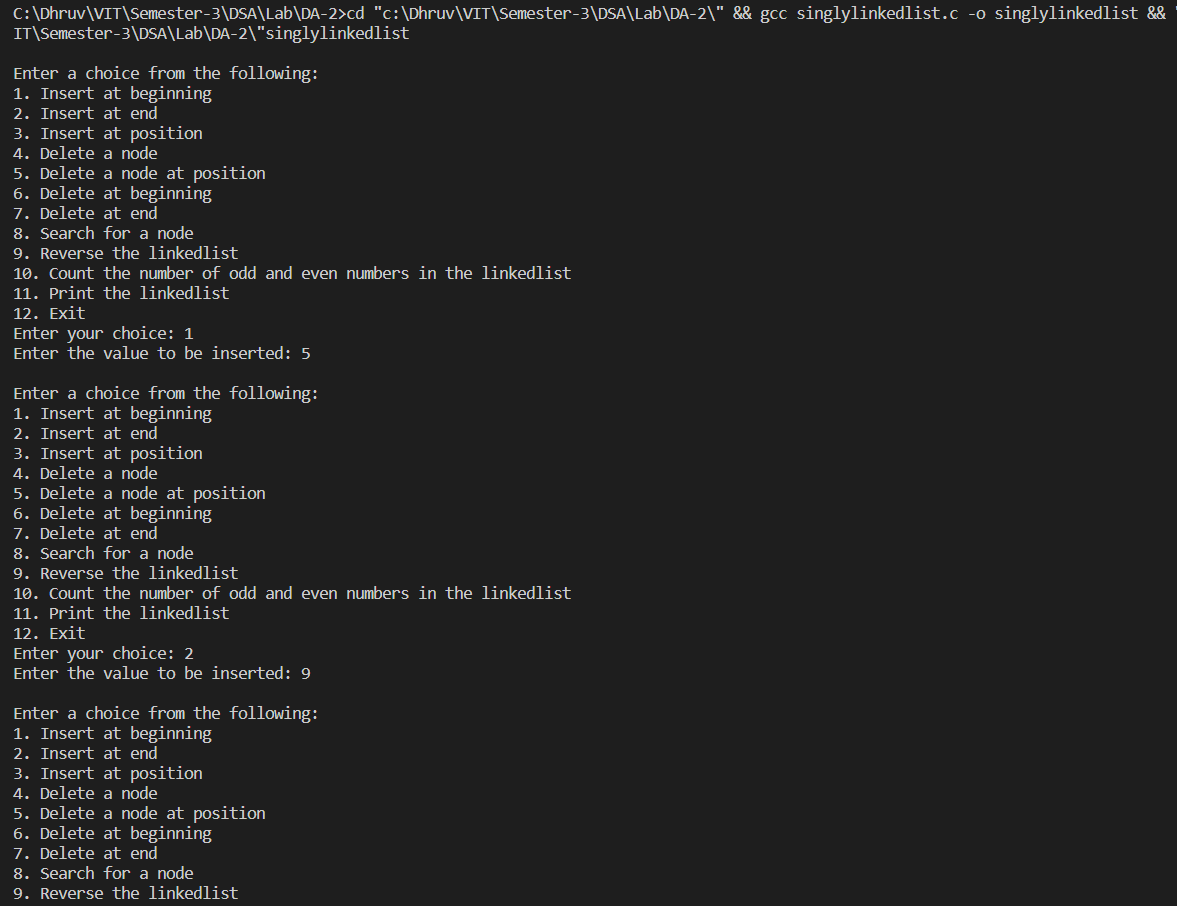
        }

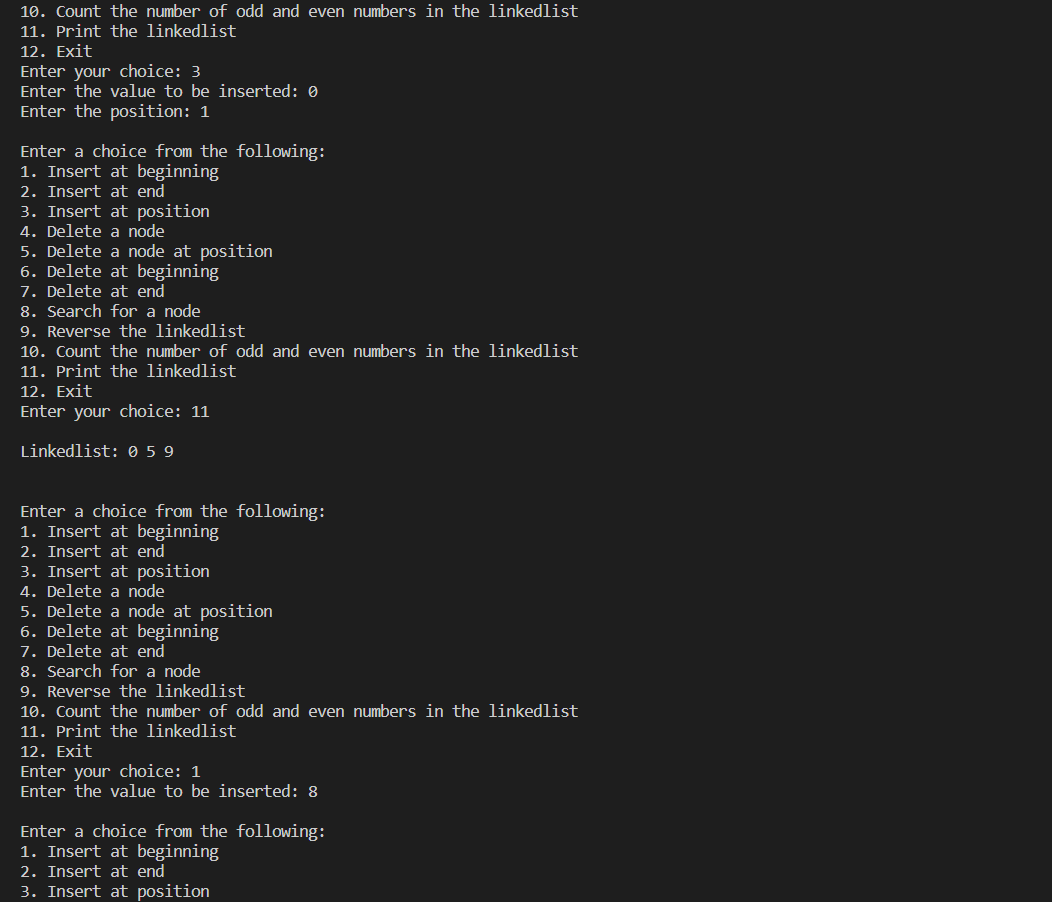
    }

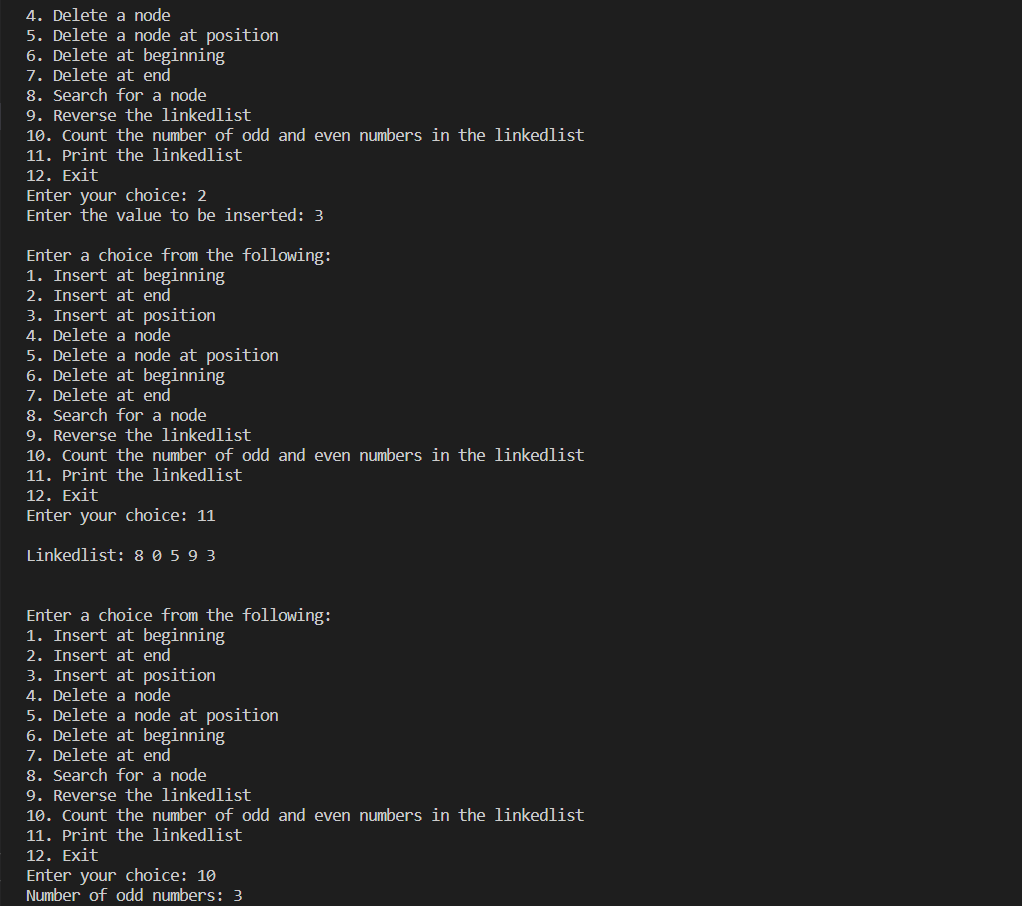
}

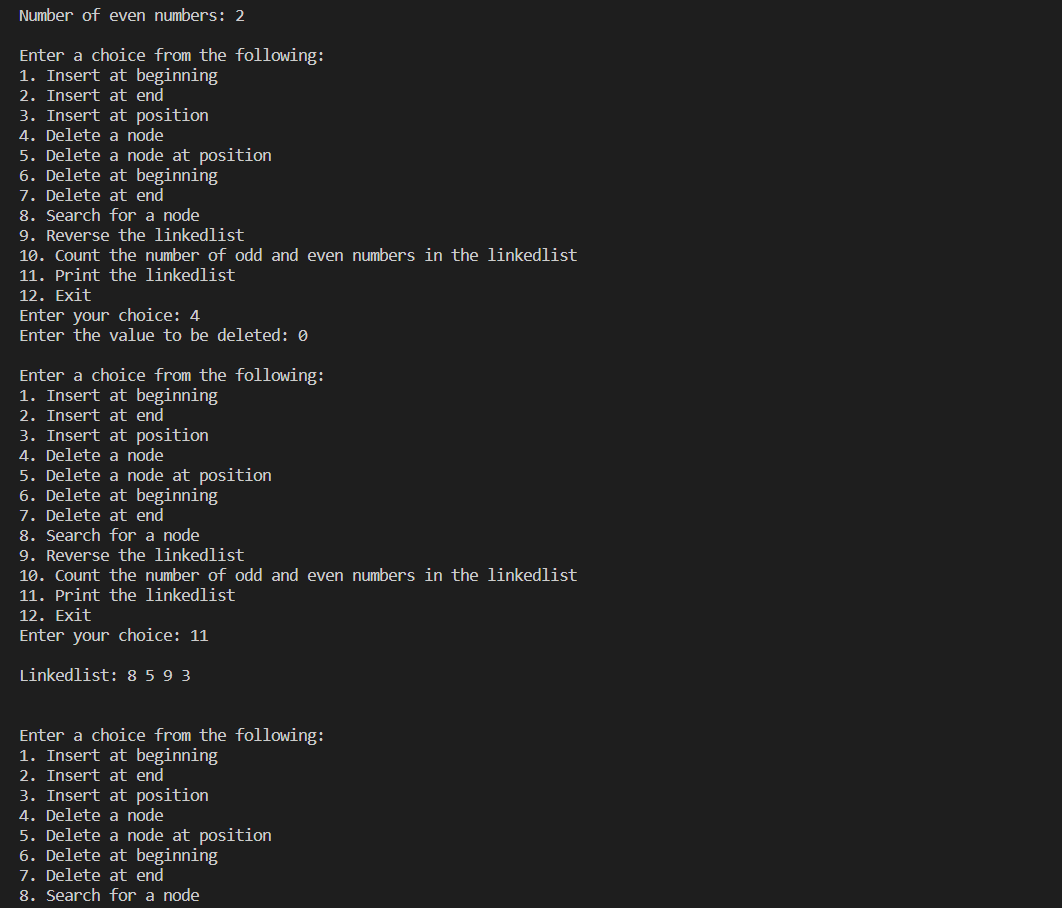
**SCREENSHOT OF CODE**

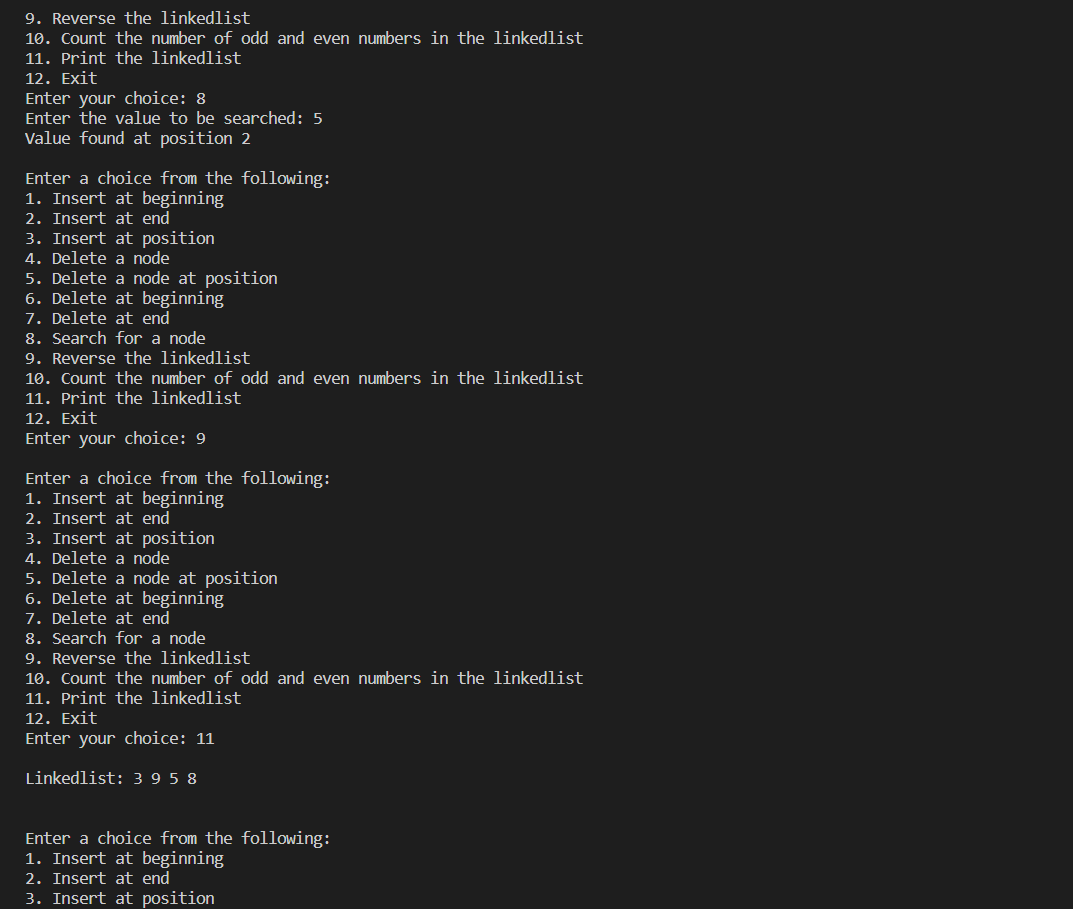
**OUTPUT**

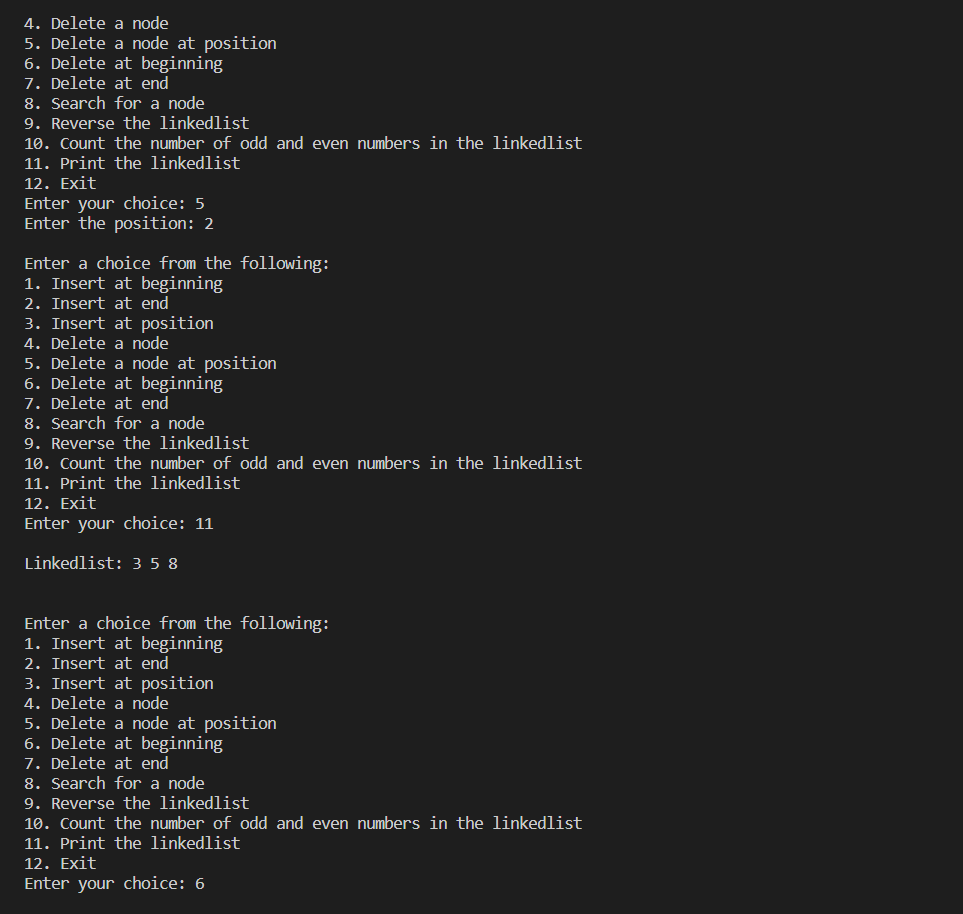
****

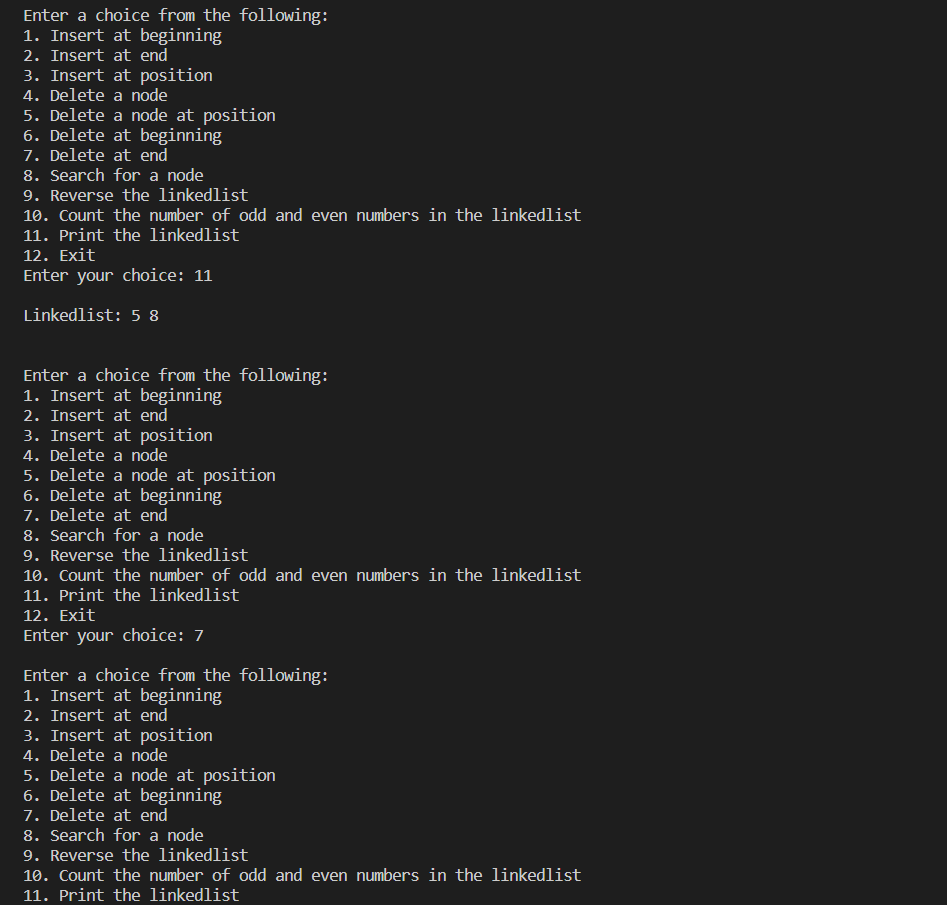


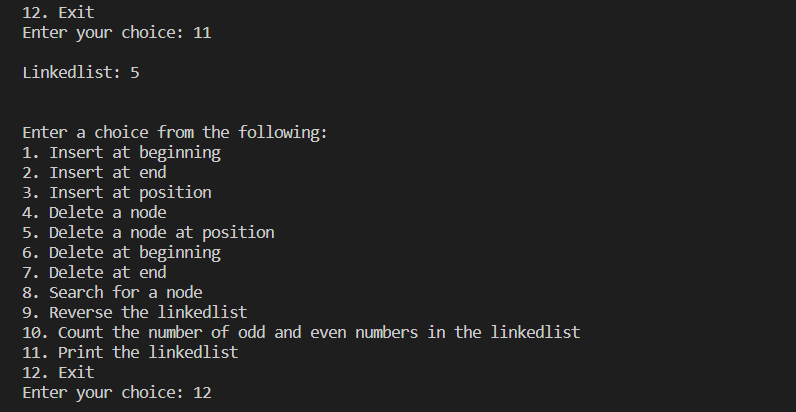












2 . Write a menu driven program to perform following functions in a doubly linked list.

i) Insertion in the beginning of the list

ii) Insertion at the end of the list

iii) Insertion in a particular location of the list

iv) Deletion based on a particular value

v) Display the contents of the list

**CODE**

// DSA

// DA-2

// Dhruv Rajeshkumar Shah

// 21BCE0611

#include <stdio.h>

#include <stdlib.h>

// Doubly linkedlist node

struct node

{

    int value;

    struct node \*next;

    struct node \*prev;

};

// Function to create a new node

struct node \*newNode(int value)

{

    struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

    new\_node->value = value;

    new\_node->next = NULL;

    new\_node->prev = NULL;

    return new\_node;

}

// Function to insert a new node at the beginning of the linkedlist

struct node \*insertAtBeginning(struct node \*head, int value)

{

    struct node \*new\_node = newNode(value);

    if (head == NULL)

    {

        head = new\_node;

        return head;

    }

    new\_node->next = head;

    head->prev = new\_node;

    head = new\_node;

    return head;

}

// Function to insert a new node at the end of the linkedlist

struct node \*insertAtEnd(struct node \*head, int value)

{

    struct node \*new\_node = newNode(value);

    if (head == NULL)

    {

        head = new\_node;

        return head;

    }

    struct node \*temp = head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    temp->next = new\_node;

    new\_node->prev = temp;

    return head;

}

// Function to insert a new node at the given position of the linkedlist

struct node \*insertAtPosition(struct node \*head, int value, int position)

{

    struct node \*new\_node = newNode(value);

    if (head == NULL)

    {

        head = new\_node;

        return head;

    }

    struct node \*temp = head;

    int count = 1;

    while (temp->next != NULL && count < position - 1)

    {

        temp = temp->next;

        count++;

    }

    new\_node->next = temp->next;

    temp->next->prev = new\_node;

    temp->next = new\_node;

    new\_node->prev = temp;

    return head;

}

// Function to delete based on value

struct node \*deleteByValue(struct node \*head, int value)

{

    if (head == NULL)

    {

        return head;

    }

    if (head->value == value)

    {

        head = head->next;

        head->prev = NULL;

        return head;

    }

    struct node \*temp = head;

    while (temp->next != NULL)

    {

        if (temp->next->value == value)

        {

            temp->next = temp->next->next;

            temp->next->prev = temp;

            return head;

        }

        temp = temp->next;

    }

    return head;

}

// Function to display the linkedlist

void display(struct node \*head)

{

    struct node \*temp = head;

    printf("\nLinked list: ");

    while (temp != NULL)

    {

        printf("%d ", temp->value);

        temp = temp->next;

    }

    printf("\n\n");

}

// Main function

int main()

{

    struct node \*head = NULL;

    int choice = 0;

    while (choice != 6)

    {

        printf("1. Insert at beginning\n");

        printf("2. Insert at end\n");

        printf("3. Insert at position\n");

        printf("4. Delete by value\n");

        printf("5. Display\n");

        printf("6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            printf("Enter the value: ");

            int val1;

            scanf("%d", &val1);

            head = insertAtBeginning(head, val1);

            break;

        case 2:

            printf("Enter the value: ");

            int val2;

            scanf("%d", &val2);

            head = insertAtEnd(head, val2);

            break;

        case 3:

            printf("Enter the value: ");

            int val3, position3;

            scanf("%d", &val3);

            printf("Enter the position: ");

            scanf("%d", &position3);

            head = insertAtPosition(head, val3, position3);

            break;

        case 4:

            printf("Enter the value: ");

            int val4;

            scanf("%d", &val4);

            head = deleteByValue(head, val4);

            break;

        case 5:

            display(head);

            break;

        case 6:

            printf("Exiting...\n");

            break;

        default:

            printf("Invalid choice\n");

            break;

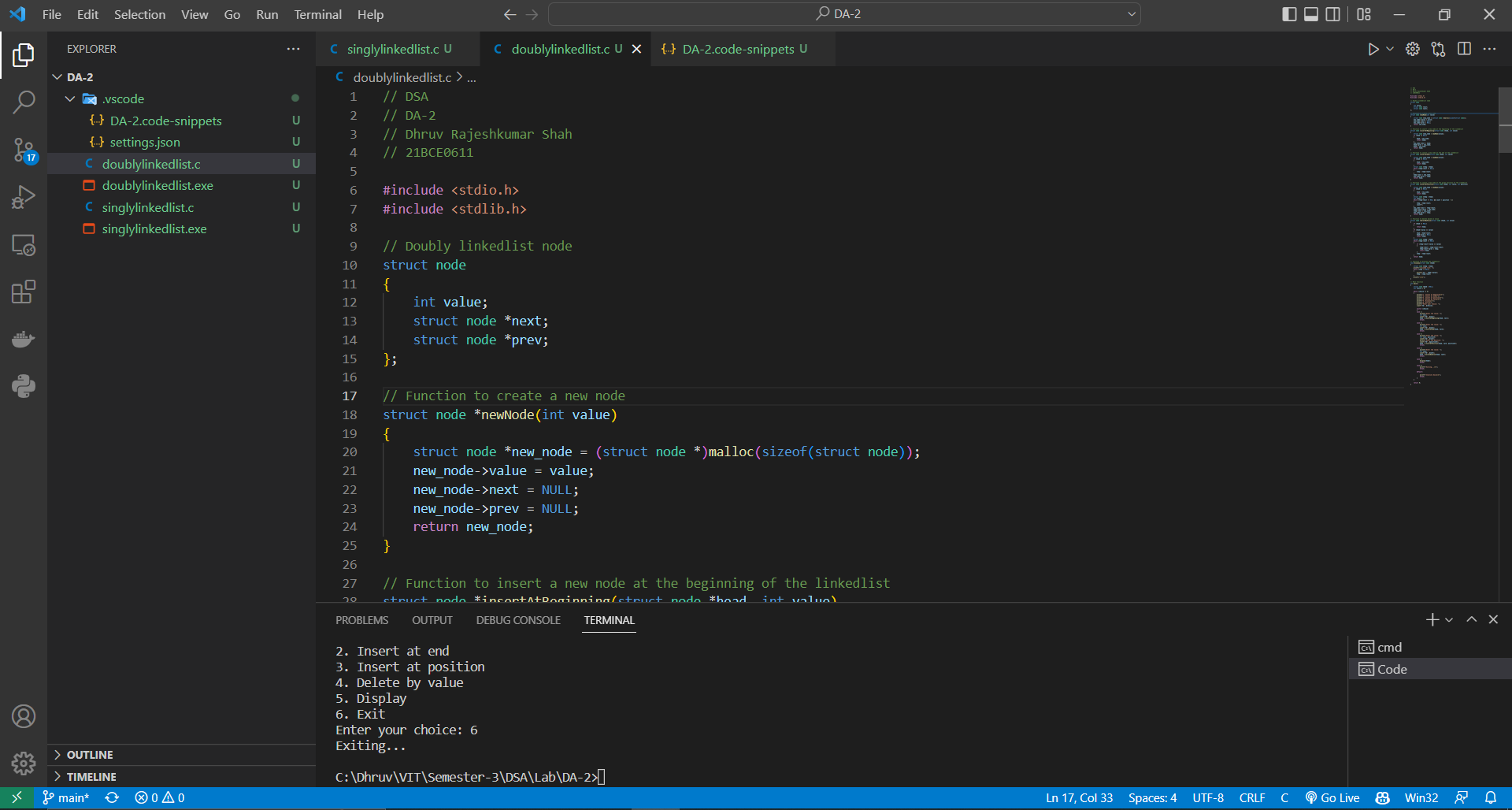
        }

    }

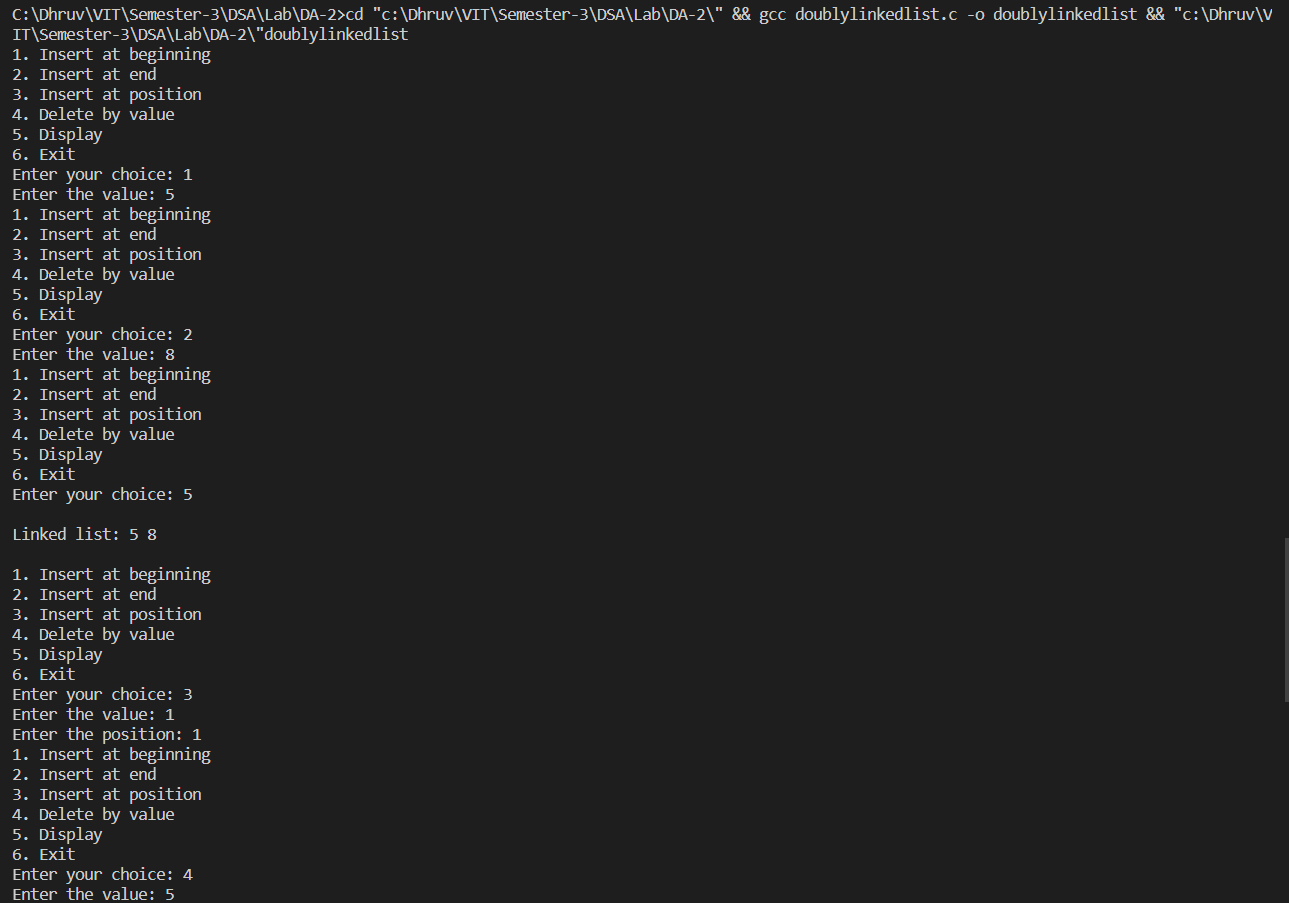
    return 0;

}

**SCREENSHOT**

****

**OUTPUTS**

****



3. Write a menu driven program to perform following functions in a circularly singly linked

list.

i) Insertion in the beginning of the list

ii) Insertion at the end of the list

iii) Deletion from the beginning of the list

iv) Deletion from the end of the list.

**CODE**

// DSA

// DA-2

// Dhruv Rajeshkumar Shah

// 21BCE0611

#include <stdio.h>

#include <stdlib.h>

// Circularly linkedlist node

struct node

{

    int value;

    struct node \*next;

};

// Function to create a new node for circularly linkedlist

struct node \*newNode(int value)

{

    struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

    new\_node->value = value;

    new\_node->next = NULL;

    return new\_node;

}

// Function to insert a new node at the beginning of the circularly linkedlist

struct node \*insertAtBeginning(struct node \*head, int value)

{

    struct node \*new\_node = newNode(value);

    if (head == NULL)

    {

        head = new\_node;

        head->next = head;

        return head;

    }

    struct node \*temp = head;

    while (temp->next != head)

    {

        temp = temp->next;

    }

    temp->next = new\_node;

    new\_node->next = head;

    head = new\_node;

    return head;

}

// Function to insert a new node at the end of the circularly linkedlist

struct node \*insertAtEnd(struct node \*head, int value)

{

    struct node \*new\_node = newNode(value);

    if (head == NULL)

    {

        head = new\_node;

        head->next = head;

        return head;

    }

    struct node \*temp = head;

    while (temp->next != head)

    {

        temp = temp->next;

    }

    temp->next = new\_node;

    new\_node->next = head;

    return head;

}

// Function to delete a node from the beginning of the circularly linkedlist

struct node \*deleteFromBeginning(struct node \*head)

{

    if (head == NULL)

    {

        printf("Linkedlist is empty\n");

        return head;

    }

    struct node \*temp = head;

    while (temp->next != head)

    {

        temp = temp->next;

    }

    temp->next = head->next;

    free(head);

    head = temp->next;

    return head;

}

// Function to delete a node from the end of the circularly linkedlist

struct node \*deleteFromEnd(struct node \*head)

{

    if (head == NULL)

    {

        printf("Linkedlist is empty\n");

        return head;

    }

    struct node \*temp = head;

    while (temp->next->next != head)

    {

        temp = temp->next;

    }

    free(temp->next);

    temp->next = head;

    return head;

}

// Function to print the circularly linkedlist

void printCircularlyLinkedList(struct node \*head)

{

    if (head == NULL)

    {

        printf("Linkedlist is empty\n");

        return;

    }

    struct node \*temp = head;

    while (temp->next != head)

    {

        printf("%d ", temp->value);

        temp = temp->next;

    }

    printf("%d ", temp->value);

    printf("\n\n");

}

// Main function

int main()

{

    struct node \*head = NULL;

    int choice = 0, value;

    while (choice != 6)

    {

        printf("1. Insert at beginning of the circularly linkedlist\n");

        printf("2. Insert at end of the circularly linkedlist\n");

        printf("3. Delete from beginning of the circularly linkedlist\n");

        printf("4. Delete from end of the circularly linkedlist\n");

        printf("5. Print the circularly linkedlist\n");

        printf("6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            printf("Enter the value: ");

            scanf("%d", &value);

            head = insertAtBeginning(head, value);

            break;

        case 2:

            printf("Enter the value: ");

            scanf("%d", &value);

            head = insertAtEnd(head, value);

            break;

        case 3:

            head = deleteFromBeginning(head);

            break;

        case 4:

            head = deleteFromEnd(head);

            break;

        case 5:

            printCircularlyLinkedList(head);

            break;

        case 6:

            printf("Exiting...\n");

            break;

        default:

            printf("Invalid choice\n");

            break;

        }

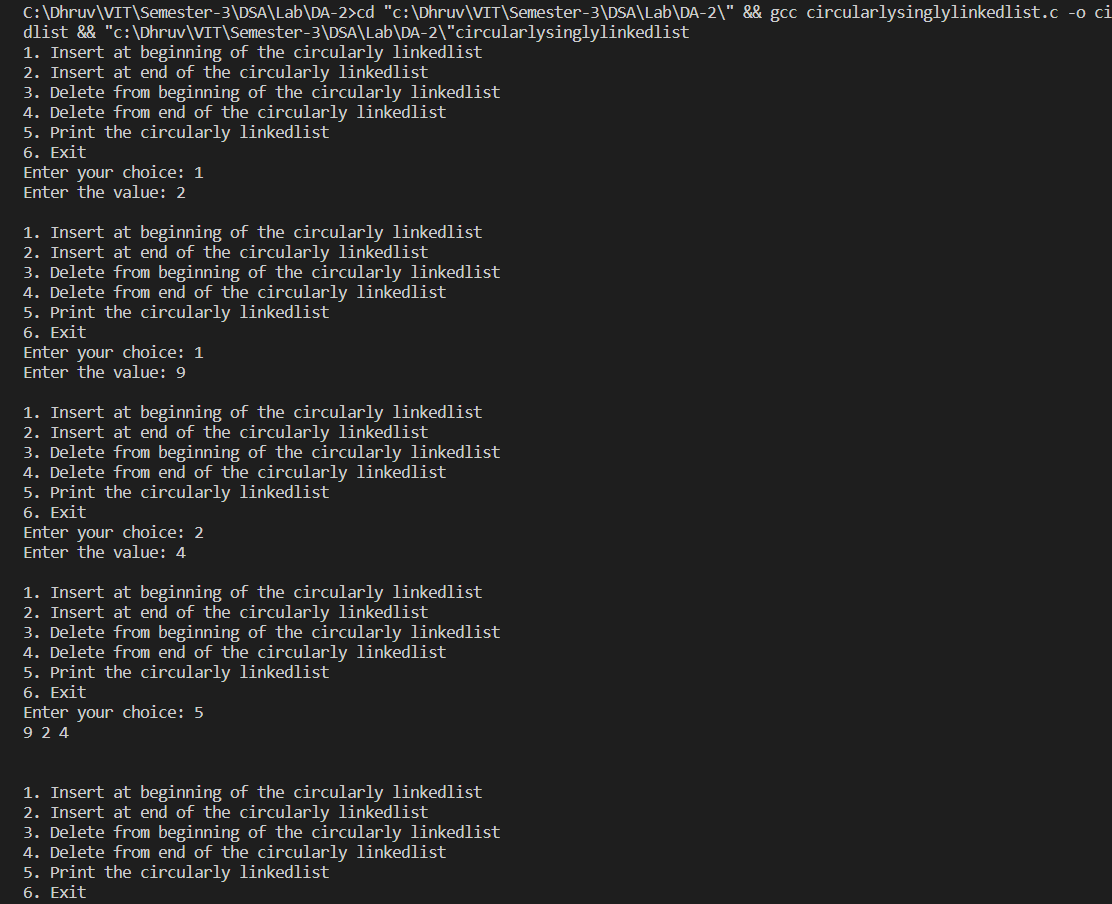
        printf("\n");

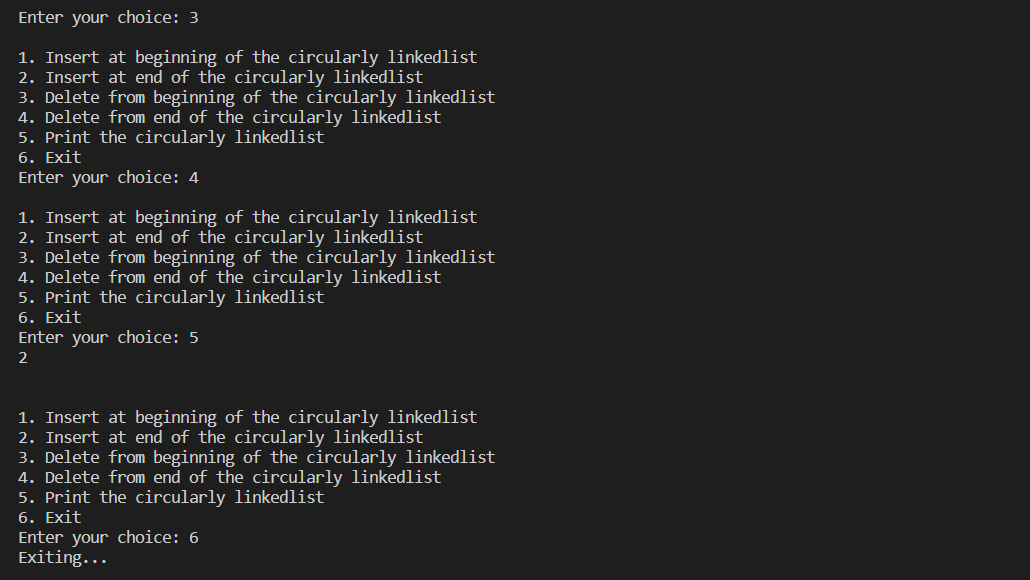
    }

    return 0;

}

**SCREENSHOT**

****



4. Create linked list to enroll the students who wish to participate for a gaming event by taking details like Name, Register No., Age, Phone number. Ensure that no more than five members are there in the list with same age. Perform insertion(), deletion() and display() operations on the Linked List

**CODE**

// DSA

// DA-2

// Dhruv Rajeshkumar Shah

// 21BCE0611

#include <stdio.h>

#include <stdlib.h>

// Student struct

struct student

{

    char name[50];

    int regNo;

    int age;

    int phone;

};

// Linked list node

struct node

{

    struct student data;

    struct node \*next;

};

// Function to create a new node

struct node \*newNode(struct student data)

{

    struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

    new\_node->data = data;

    new\_node->next = NULL;

    return new\_node;

}

// Function to count number of students with same age

int check\_age(struct node \*head, int age)

{

    struct node \*new\_node;

    int count = 0;

    new\_node = head;

    while (new\_node != NULL)

    {

        if (new\_node->data.age == age)

        {

            count++;

        }

        new\_node = new\_node->next;

    }

    return count;

}

// Function to insert a new node to linkedlist

struct node \*insert(struct node \*head, struct student data)

{

    if (check\_age(head, data.age) >= 5)

    {

        printf("Can't have more than 5 students of same age!\n\n");

        return head;

    }

    struct node \*new\_node = newNode(data);

    if (head == NULL)

    {

        head = new\_node;

        return head;

    }

    struct node \*temp = head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    temp->next = new\_node;

    return head;

}

// Function to delete a not at the beginning of the linkedlist

struct node \*delete (struct node \*head, int regNo)

{

    if (head == NULL)

    {

        printf("Linkedlist is empty");

        return head;

    }

    struct node \*temp = head;

    while (temp->next != NULL)

    {

        if (temp->next->data.regNo == regNo)

        {

            struct node \*temp2 = temp->next;

            temp->next = temp->next->next;

            free(temp2);

            return head;

        }

        temp = temp->next;

    }

    printf("Student not found");

    return head;

}

// Function to display the linkedlist

void display(struct node \*head)

{

    if (head == NULL)

    {

        printf("Linkedlist is empty");

        return;

    }

    struct node \*temp = head;

    while (temp != NULL)

    {

        printf("Name: %s, ", temp->data.name);

        printf("RegNo: %d, ", temp->data.regNo);

        printf("Age: %d, ", temp->data.age);

        printf("Phone: %d", temp->data.phone);

        temp = temp->next;

        printf("\n");

    }

    printf("\n\n");

}

// Main function

int main()

{

    struct node \*head = NULL;

    struct student data;

    int choice = 0;

    while (choice != 4)

    {

        printf("\n1. Insert student details to linkedlist\n");

        printf("2. Delete student details from linkedlist\n");

        printf("3. Display the linkedlist\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            printf("Enter student details: \n");

            printf("Name: ");

            scanf("%s", data.name);

            printf("RegNo: ");

            scanf("%d", &data.regNo);

            printf("Age: ");

            scanf("%d", &data.age);

            printf("Phone: ");

            scanf("%d", &data.phone);

            head = insert(head, data);

            break;

        case 2:

            printf("Enter the regNo of the student to be deleted: ");

            scanf("%d", &data.regNo);

            head = delete (head, data.regNo);

            break;

        case 3:

            display(head);

            break;

        case 4:

            break;

        default:

            printf("Invalid choice");

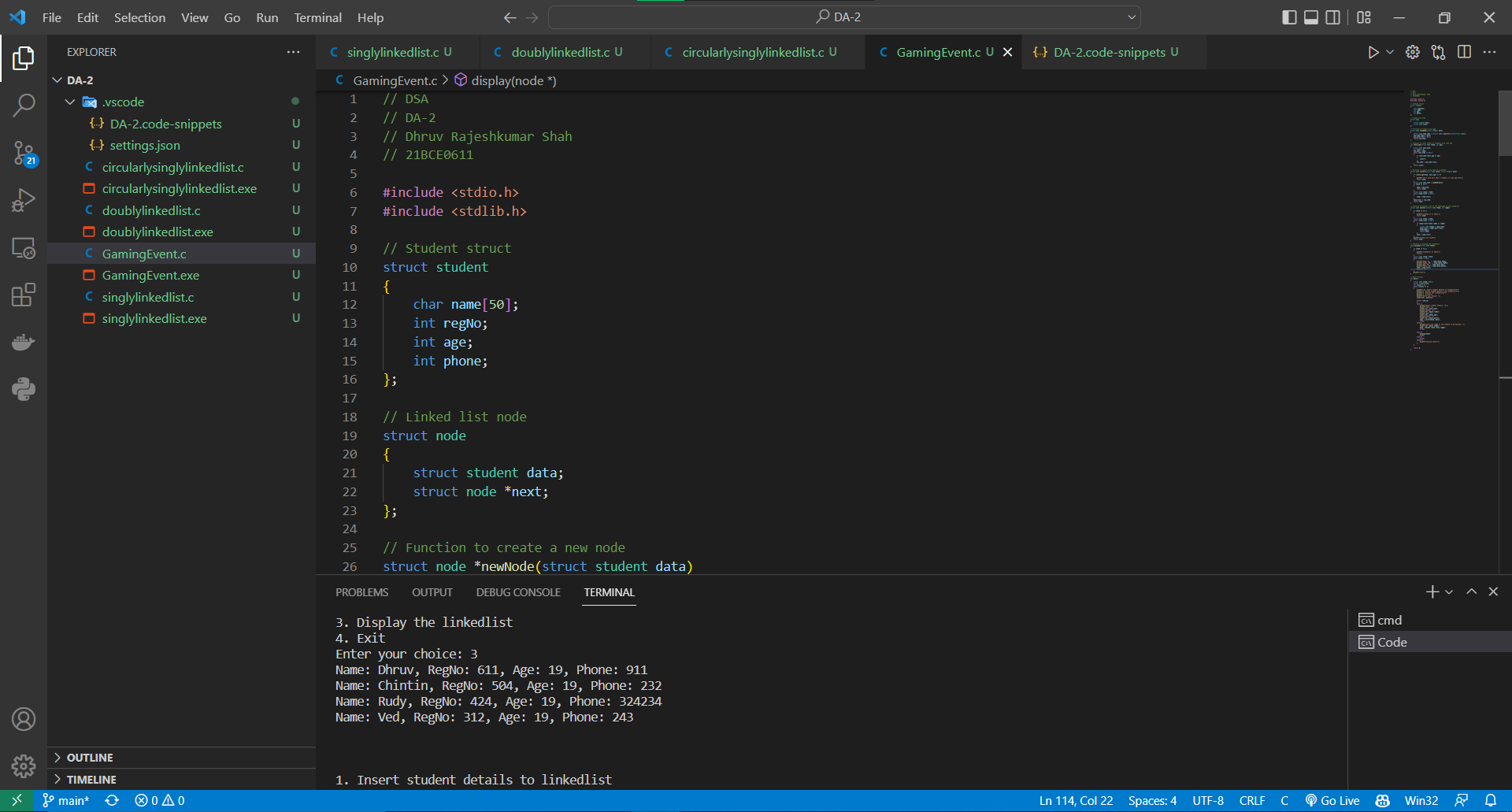
        }

    }

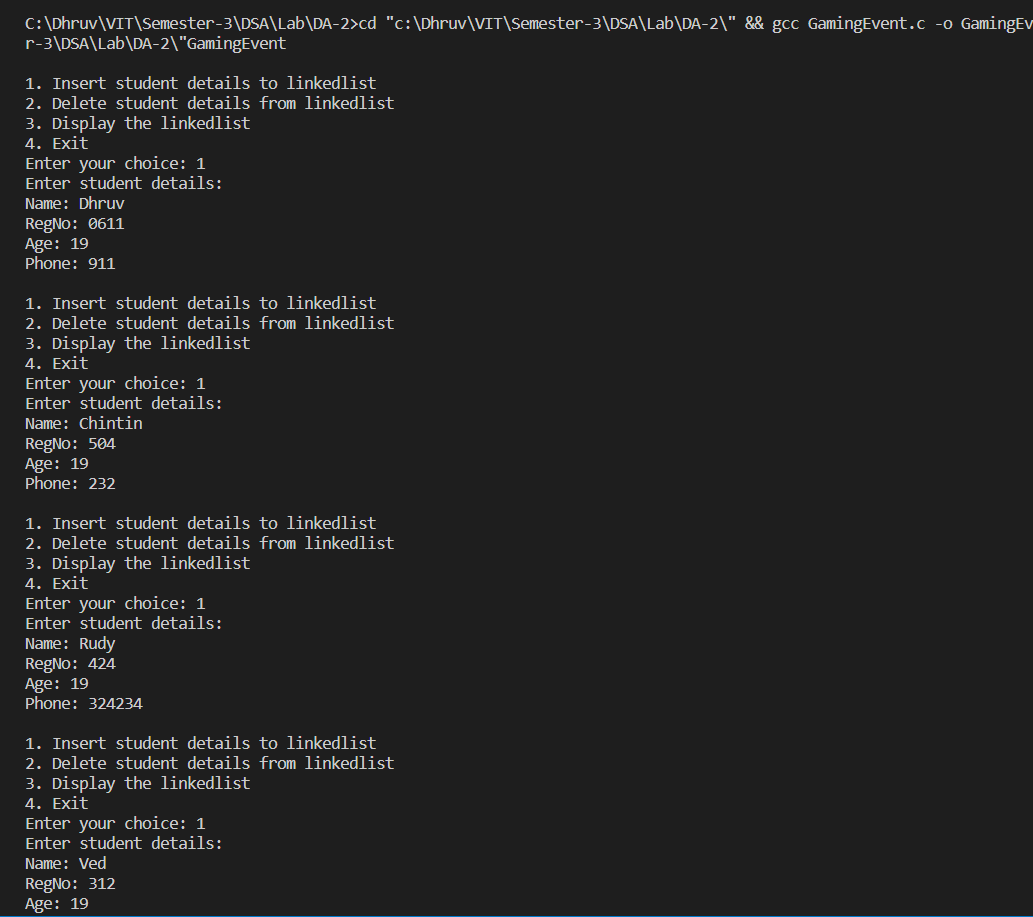
    return 0;

}

**SCREENSHOT**

****

**OUTPUT**

****

