DATA STRUCTURES AND ALGORITHMS

CYCLESHEET-2

NAME: ANUBHAV JAIN
REG.NO: 22BIT0210
FACULTY: PROF VIJAYAN.E

S.NO	Name of Program	Page No.
1.	Write a C program to implement the following operations on singly linked list. i. creation ii. insertion all cases iii. deletion all cases iv. display	2-4
2.	Write a C program to implement the following operations on doubly linked list. i. creation ii. insertion all cases iii. deletion all cases iv. display	5-7
3.	Write a C program to implement the following operations on singly circular linked list. i. creation ii. insertion all cases iii. deletion all cases iv. display	8-11
4.	Write a C program to implement the following operations on doubly circular linked list. i. creation ii. insertion all cases iii. deletion all cases iv. display	12-13

5.	Write a C program to implement stack using linked list.	14-16
6.	Write a C program to implement queue using linked list.	17-18
7.	Write a C program to implement polynomial addition using linked list.	19-22
8.	Given two sorted lists L1 and L2 write a program to merge the two lists in sorted order.	23-26
9.	Given two list L1 and L2 write a C program to find the intersection of two list.	27-29
10.	Assume FLAMES game that tests for relationship has to be implemented using a dynamic structure. The letters in the FLAMES stand for Friends, Love, Affection, Marriage, Enmity and Sister. Initially store the individual letters of the word 'flames' in the nodes of the dynamic structure. Given the count of the number of uncommon letters in the two names 'n', write a program to delete every nth node in it, till it is left with a single node. If the end of the dynamic structure is reached while counting, resume the counting from the beginning. Display the letter that still remains and the corresponding relationship Eg., If Ajay and Jack are the two names, there are 4 uncommon letters in these. So delete 4th node in the first iteration and for the next iteration start counting from the node following the deleted node.	30-32

Data Structures and Algorithms

Cyclesheet-1

Name: Anubhav Jain

Reg.No:22BIT0210

1.Write a C program to implement the following operations on singly linked list.

i. creation

ii.insertion all cases

iii. deletion all cases

iv. display

Source code:-

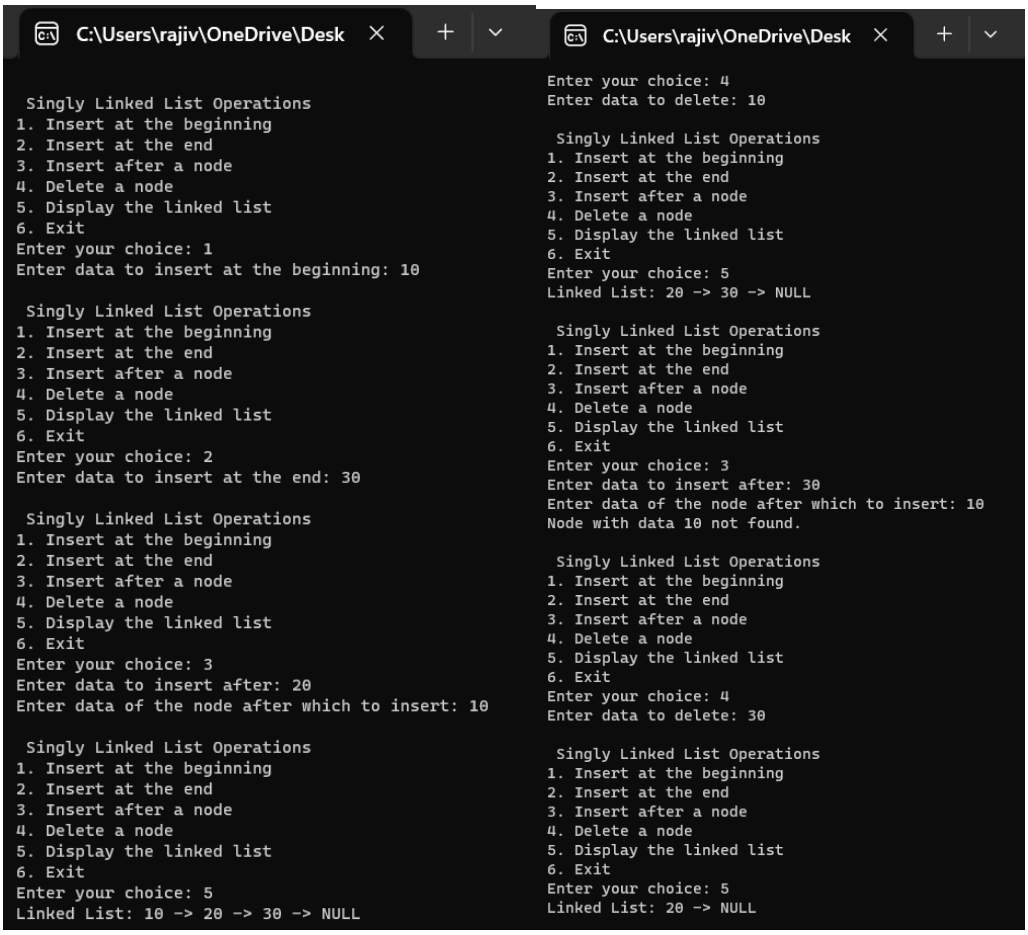
```c
#include <stdio.h>
#include <stdlib.h>

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}
```

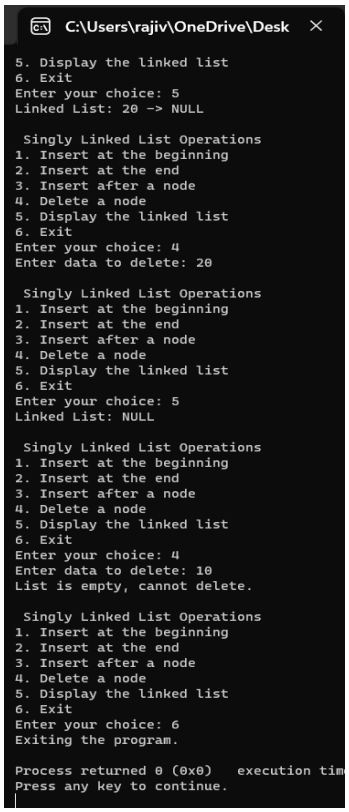
```c
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}

void insertAfter(struct Node* prevNode, int data) {
    if (prevNode == NULL) {
        printf("Previous node cannot be NULL.\n");
        return;
    }
    struct Node* newNode = createNode(data);
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}

void deleteNode(struct Node** head, int data) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }
    struct Node* current = *head;
    struct Node* prev = NULL;

    if (current->data == data) {
        *head = current->next;
        free(current);
        return;
    }

    while (current != NULL && current->data != data) {
        prev = current;
        current = current->next;
    }

    if (current == NULL) {
        printf("Node with data %d not found.\n", data);
```

```
        return;
    }

    prev->next = current->next;
    free(current);
}
void display(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data;

    while (1) {
        printf("\n Singly Linked List Operations\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert at the end\n");
        printf("3. Insert after a node\n");
        printf("4. Delete a node\n");
        printf("5. Display the linked list\n");
        printf("6. 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 at the
end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                printf("Enter data to insert after: ");
```

```
                scanf("%d", &data);
                printf("Enter data of the node after
which to insert: ");
                int afterData;
                scanf("%d", &afterData);
                struct Node* temp = head;
                while (temp != NULL && temp->data !=
afterData) {
                    temp = temp->next;
                }
                if (temp == NULL) {
                    printf("Node with data %d not
found.\n", afterData);
                } else {
                    insertAfter(temp, data);
                }
                break;
            case 4:
                printf("Enter data to delete: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 5:
                printf("Linked List: ");
                display(head);
                break;
            case 6:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice, please try
again.\n");
        }
    }

    return 0;
}
```

Output:-


C:\Users\rajiv\OneDrive\Desk
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 1
Enter data to insert at the beginning: 10
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 2
Enter data to insert at the end: 30
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 3
Enter data to insert after: 20
Enter data of the node after which to insert: 10
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 5
Linked List: 10 -> 20 -> 30 -> NULL
C:\Users\rajiv\OneDrive\Desk
Enter your choice: 4
Enter data to delete: 10
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 5
Linked List: 20 -> 30 -> NULL
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 3
Enter data to insert after: 30
Enter data of the node after which to insert: 10
Node with data 10 not found.
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 4
Enter data to delete: 30
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 5
Linked List: 20 -> NULL


C:\Users\rajiv\OneDrive\Desk
5. Display the linked list
6. Exit
Enter your choice: 5
Linked List: 20 -> NULL
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 4
Enter data to delete: 20
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 5
Linked List: NULL
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 4
Enter data to delete: 10
List is empty, cannot delete.
Singly Linked List Operations
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list
6. Exit
Enter your choice: 6
Exiting the program.
Process returned 0 (0x0) execution tim
Press any key to continue.

2. Write a C program to implement the following operations on doubly linked list.

i. creation

ii.insertion all cases

iii. deletion all cases

iv. display

Source code:-

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    newNode->next = *head;
    (*head)->prev = newNode;
    *head = newNode;
}
void insertAtEnd(struct Node** head, int data) {
```

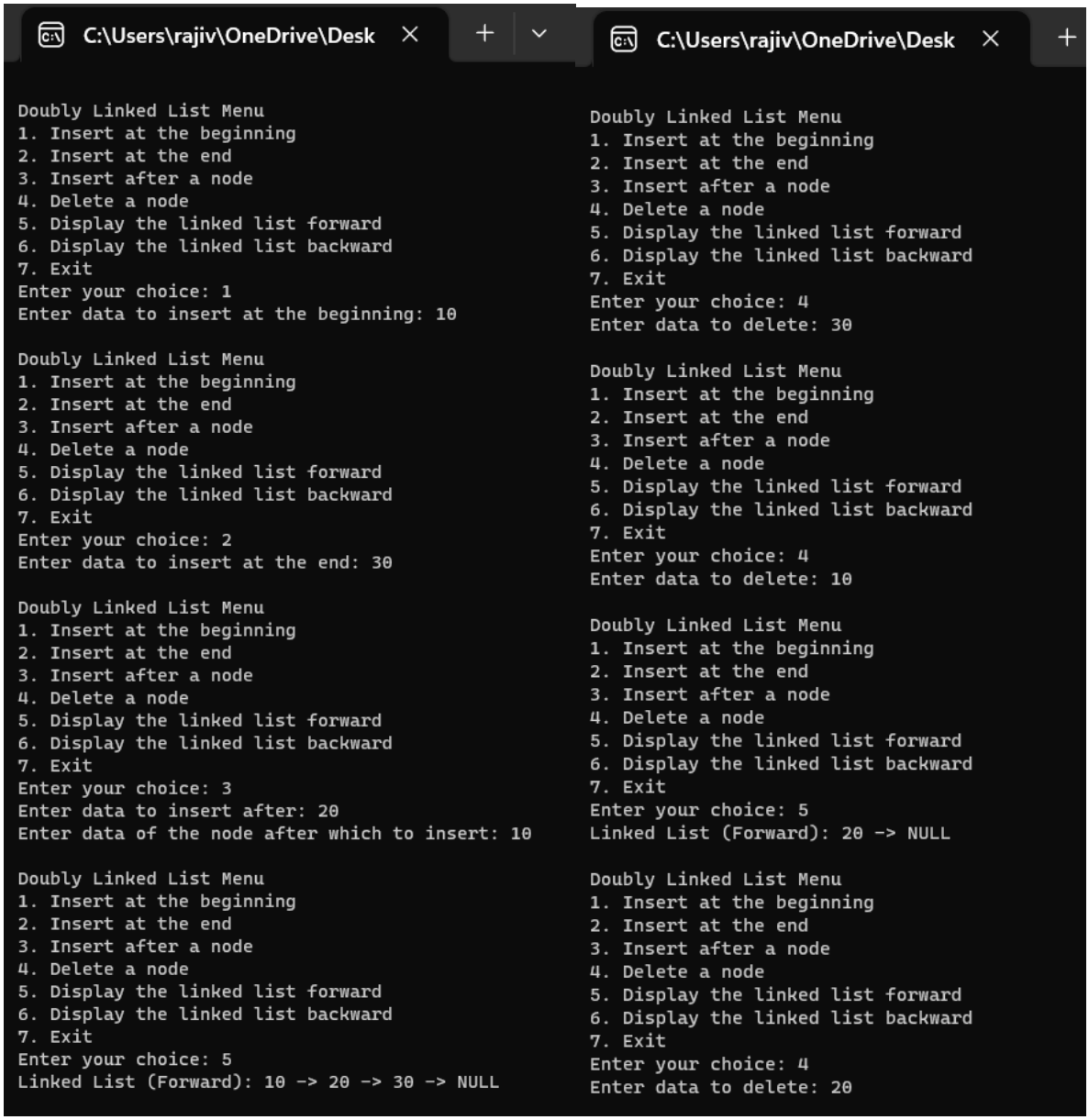
```
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
}
void insertAfter(struct Node* prevNode, int data) {
    if (prevNode == NULL) {
        printf("Previous node cannot be NULL.\n");
        return;
    }
    struct Node* newNode = createNode(data);
    newNode->next = prevNode->next;
    newNode->prev = prevNode;
    if (prevNode->next != NULL) {
        prevNode->next->prev = newNode;
    }
    prevNode->next = newNode;
}
void deleteNode(struct Node** head, int data) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }
    struct Node* current = *head;
    while (current != NULL && current->data != data) {
        current = current->next;
    }
    if (current == NULL) {
        printf("Node with data %d not found.\n", data);
        return;
    }
    if (current->prev != NULL) {
        current->prev->next = current->next;
    }
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    if (*head == current) {
```

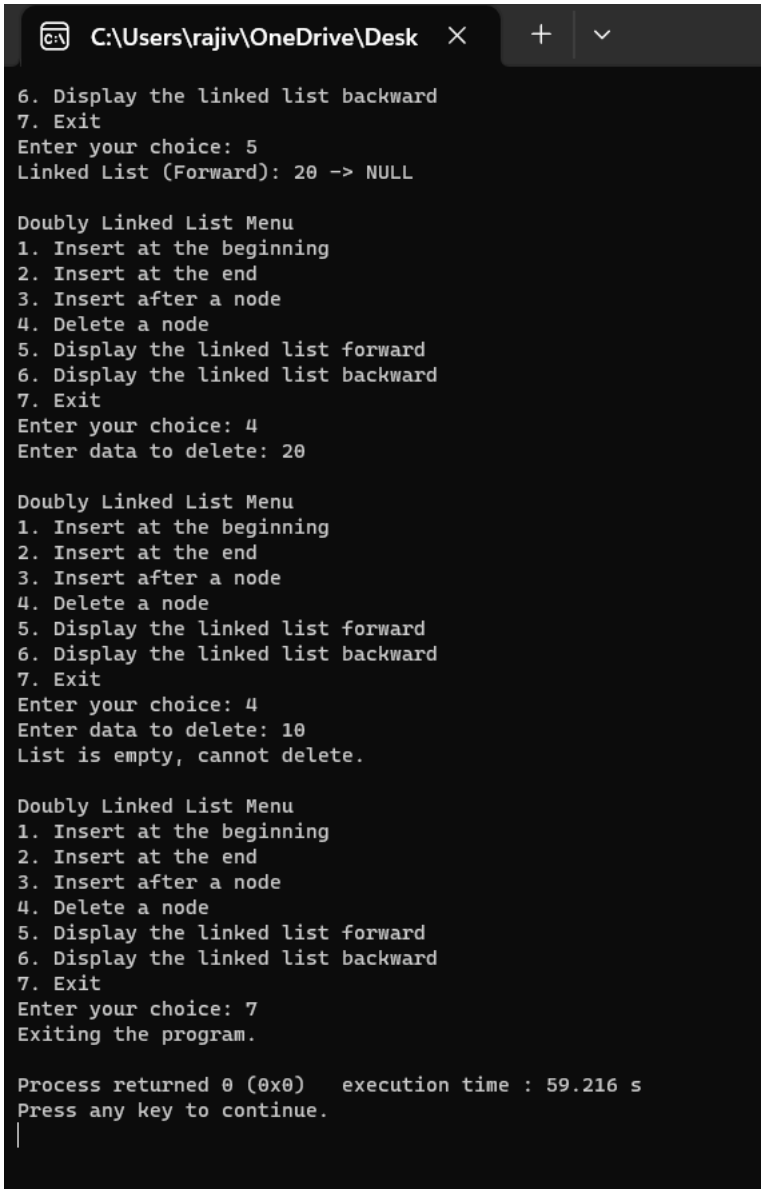
```
        *head = current->next;
    }
    free(current);
}
void displayForward(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
void displayBackward(struct Node* head) {
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    printf("NULL ");
    while (current != NULL) {
        printf("-> %d ", current->data);
        current = current->prev;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data;

    while (1) {
        printf("\nDoubly Linked List Menu\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert at the end\n");
        printf("3. Insert after a node\n");
        printf("4. Delete a node\n");
        printf("5. Display the linked list forward\n");
        printf("6. Display the linked list
backward\n");
        printf("7. 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 at the
end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                printf("Enter data to insert after: ");
                scanf("%d", &data);
                printf("Enter data of the node after
which to insert: ");
                int afterData;
                scanf("%d", &afterData);
                struct Node* temp = head;
                while (temp != NULL && temp->data !=
afterData) {
                    temp = temp->next;
                }
                if (temp == NULL) {
                    printf("Node with data %d not
found.\n", afterData);
                } else {
                    insertAfter(temp, data);
                }
                break;
            case 4:
                printf("Enter data to delete: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 5:
                printf("Linked List (Forward): ");
                displayForward(head);
                break;
            case 6:
                printf("Linked List (Backward): ");
                displayBackward(head);
                break;
            case 7:
                printf("Exiting the program.\n");
```

```
                exit(0);
            default:
                printf("Invalid choice, please try
again.\n");
        }
    }

    return 0;
}
```

Output:-




C:\Users\rajiv\OneDrive\Desk
6. Display the linked list backward
7. Exit
Enter your choice: 5
Linked List (Forward): 20 -> NULL
Doubly Linked List Menu
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list forward
6. Display the linked list backward
7. Exit
Enter your choice: 4
Enter data to delete: 20
Doubly Linked List Menu
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list forward
6. Display the linked list backward
7. Exit
Enter your choice: 4
Enter data to delete: 10
List is empty, cannot delete.
Doubly Linked List Menu
1. Insert at the beginning
2. Insert at the end
3. Insert after a node
4. Delete a node
5. Display the linked list forward
6. Display the linked list backward
7. Exit
Enter your choice: 7
Exiting the program.
Process returned 0 (0x0) execution time : 59.216 s
Press any key to continue.

3. Write a C program to implement the following operations on singly circular linked list.

i. creation

ii.insertion all cases

iii. deletion all cases

iv. display

Source code:-

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
struct Node* insertAtBeginning(struct Node* head, int
data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        newNode->next = newNode;
        return newNode;
    }
    newNode->next = head->next;
    head->next = newNode;
    return head;
}

struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
```

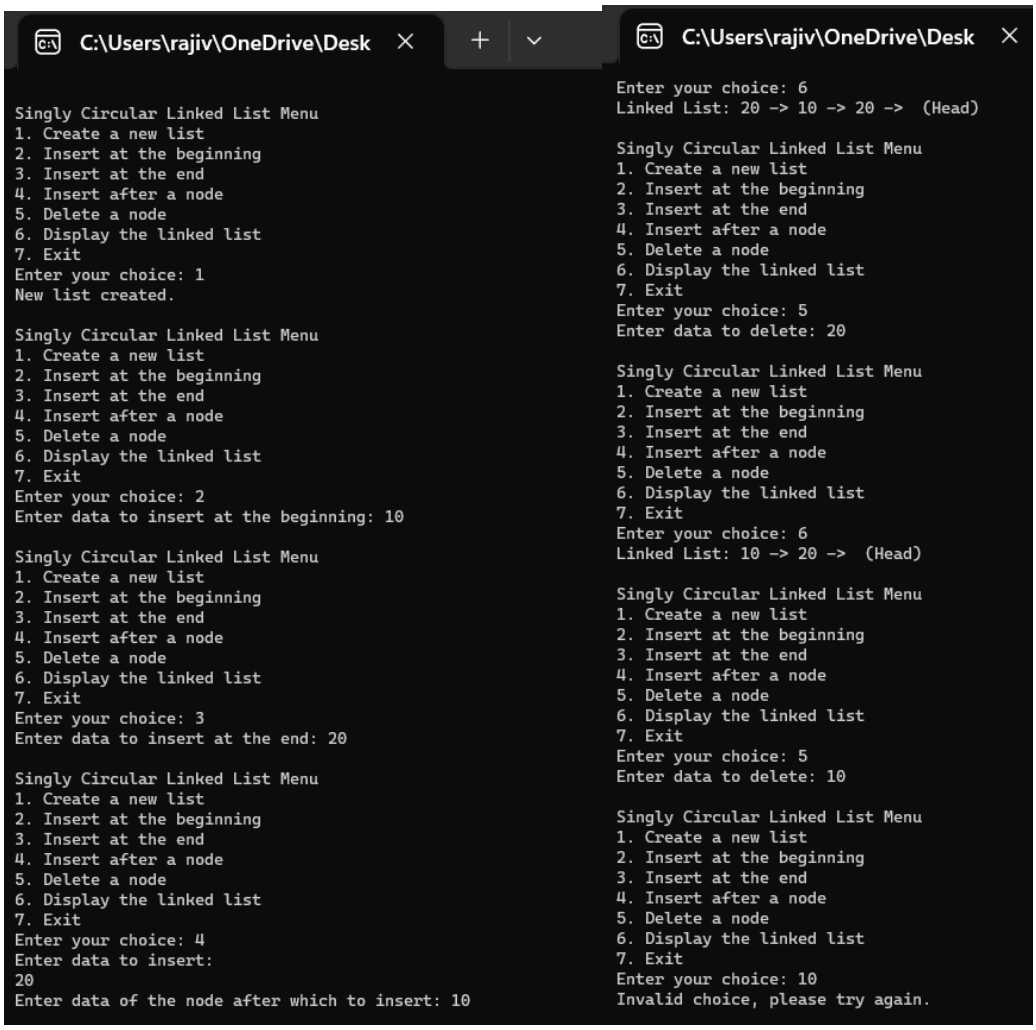
```
        newNode->next = newNode;
        return newNode;
    }
    newNode->next = head->next;
    head->next = newNode;
    return newNode;
}
struct Node* insertAfter(struct Node* head, int
afterData, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        printf("List is empty, cannot insert
after.\n");
        return head;
    }
    struct Node* current = head;
    do {
        if (current->data == afterData) {
            newNode->next = current->next;
            current->next = newNode;
            return head;
        }
        current = current->next;
    } while (current != head);
    printf("Node with data %d not found.\n",
afterData);
    return head;
}
struct Node* deleteNode(struct Node* head, int data) {
    if (head == NULL) {
        printf("List is empty, cannot delete.\n");
        return head;
    }
    struct Node* current = head;
    struct Node* prev = NULL;

    do {
        if (current->data == data) {
            if (current == head) {
                head = head->next;
                struct Node* temp = current->next;
                while (temp->next != current) {
                    temp = temp->next;
                }
                temp->next = head;
```

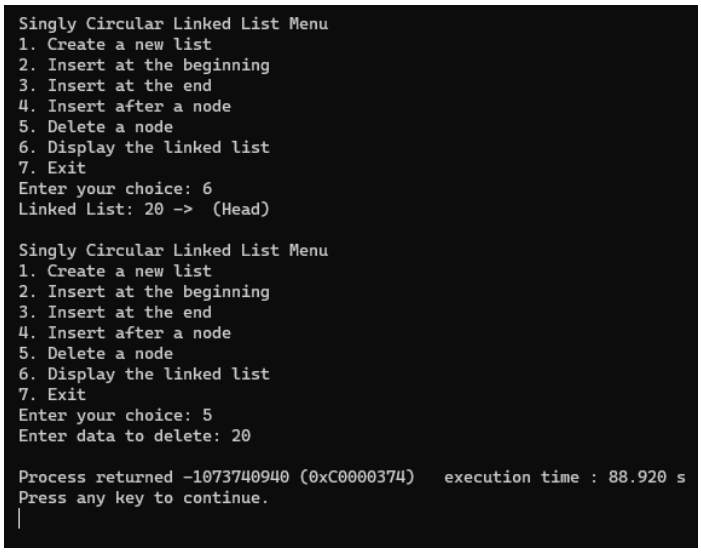
```
                free(current);
                return head;
            }
            prev->next = current->next;
            free(current);
            return head;
        }
        prev = current;
        current = current->next;
    } while (current != head);

    printf("Node with data %d not found.\n", data);
    return head;
}

// Function to display the singly circular linked list
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* current = head;
    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != head);
    printf(" (Head)\n");
}

int main() {
    struct Node* head = NULL;

    while (1) {
        printf("\nSingly Circular Linked List Menu\n");
        printf("1. Create a new list\n");
        printf("2. Insert at the beginning\n");
        printf("3. Insert at the end\n");
        printf("4. Insert after a node\n");
        printf("5. Delete a node\n");
        printf("6. Display the linked list\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");

        int choice, data, afterData;
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                head = NULL;
                printf("New list created.\n");
                break;
            case 2:
                printf("Enter data to insert at the
beginning: ");
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                break;
            case 3:
                printf("Enter data to insert at the
end: ");
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                break;
            case 4:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter data of the node after
which to insert: ");
                scanf("%d", &afterData);
                head = insertAfter(head, afterData,
data);
                break;
            case 5:
                printf("Enter data to delete: ");
                scanf("%d", &data);
                head = deleteNode(head, data);
                break;
            case 6:
                printf("Linked List: ");
                display(head);
                break;
            case 7:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice, please try
again.\n");
        }
    }
```

```
    return 0;
}
```

OUTPUT:





4. Write a C program to implement the following operations on doubly circular linked list.

i. creation

ii.insertion all cases

iii. deletion all cases

iv. display

SOURCE CODE:

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

struct Node* insertAtBeginning(struct Node* head, int
data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        newNode->next = newNode;
        newNode->prev = newNode;
        return newNode;
    }
    newNode->next = head;
    newNode->prev = head->prev;
    head->prev->next = newNode;
    head->prev = newNode;
```

```
    return newNode;
}

struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        newNode->next = newNode; // Point to itself to
create a circular link
        newNode->prev = newNode;
        return newNode;
    }
    newNode->next = head;
    newNode->prev = head->prev;
    head->prev->next = newNode;
    head->prev = newNode;
    return head;
}

struct Node* insertAfter(struct Node* head, int
afterData, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        printf("List is empty, cannot insert
after.\n");
        return head;
    }
    struct Node* current = head;
    do {
        if (current->data == afterData) {
            newNode->next = current->next;
            newNode->prev = current;
            current->next->prev = newNode;
            current->next = newNode;
            return head;
        }
        current = current->next;
    } while (current != head);
    printf("Node with data %d not found.\n",
afterData);
    return head;
}

struct Node* deleteNode(struct Node* head, int data) {
    if (head == NULL) {
        printf("List is empty, cannot delete.\n");
```

```
        return head;
    }
    struct Node* current = head;
    do {
        if (current->data == data) {
            if (current == head) {
                head = head->next;
                head->prev = current->prev;
                current->prev->next = head;
                free(current);
                return head;
            }
            current->prev->next = current->next;
            current->next->prev = current->prev;
            free(current);
            return head;
        }
        current = current->next;
    } while (current != head);

    printf("Node with data %d not found.\n", data);
    return head;
}

void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* current = head;
    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != head);
    printf(" (Head)\n");
}

int main() {
    struct Node* head = NULL;

    while (1) {
        printf("\nDoubly Circular Linked List Menu\n");
        printf("1. Create a new list\n");
        printf("2. Insert at the beginning\n");
        printf("3. Insert at the end\n");
```

```
        printf("4. Insert after a node\n");
        printf("5. Delete a node\n");
        printf("6. Display the linked list\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");

        int choice, data, afterData;
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                head = NULL;
                printf("New list created.\n");
                break;
            case 2:
                printf("Enter data to insert at the
beginning: ");
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                break;
            case 3:
                printf("Enter data to insert at the
end: ");
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                break;
            case 4:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter data of the node after
which to insert: ");
                scanf("%d", &afterData);
                head = insertAfter(head, afterData,
data);
                break;
            case 5:
                printf("Enter data to delete: ");
                scanf("%d", &data);
                head = deleteNode(head, data);
                break;
            case 6:
                printf("Linked List: ");
                display(head);
                break;
            case 7:
```

```
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice, please try
again.\n");
        }
    }

    return 0;
}
```

OUTPUT:

5. Write a C program to implement stack using linked list.

SOURCE CODE:

```
//stack using linked list
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
void push(void);
void pop(void);
void display(void);
struct node
{
int data;
struct node *next;
};struct node
*nw,*top,*temp;
void main()
{
int choice;
nw=((struct node*)malloc(sizeof(struct node)));
nw->data=0;
nw->next=NULL;
top=nw;
do
{
printf("\n1.push\n2.pop\n3.display");
printf("\nEnter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
}}while(choice<4);
getch();
}
void push()
```

```
{
int item;
printf("\nenter theitem");
scanf("%d",&item);
if(top->data==0)
top->data=item;
else
{
nw=((struct node*)malloc(sizeof(struct node)));
nw->data=item;
nw->next=top;
top=nw;
}}
void pop()
{
if(top->data==0)
printf("\nstack underflow");
else
top=top->next;
}
void display()
{
if(top->data==0)
printf("\nstack empty");
else
{
temp=top;
while(temp!=NULL)
{
printf("\t%d",temp->data)
;
temp=temp->next;
}}}
```

OUTPUT:


C:\Users\rajiv\OneDrive\Desk
1.push
2.pop
3.display
Enter your choice:3
stack empty
1.push
2.pop
3.display
Enter your choice:1
enter theitem10
1.push
2.pop
3.display
Enter your choice:1
enter theitem20
1.push
2.pop
3.display
Enter your choice:3
20 10
1.push
2.pop
3.display
Enter your choice:2
1.push
2.pop
3.display
Enter your choice:3
10
1.push
2.pop
C:\Users\rajiv\OneDrive\Desk
enter theitem20
1.push
2.pop
3.display
Enter your choice:3
20 10
1.push
2.pop
3.display
Enter your choice:2
1.push
2.pop
3.display
Enter your choice:3
10
1.push
2.pop
3.display
Enter your choice:2
1.push
2.pop
3.display
Enter your choice:3
Process returned -1073741819 (0xC0000005) execution time : 22.849 s
Press any key to continue.

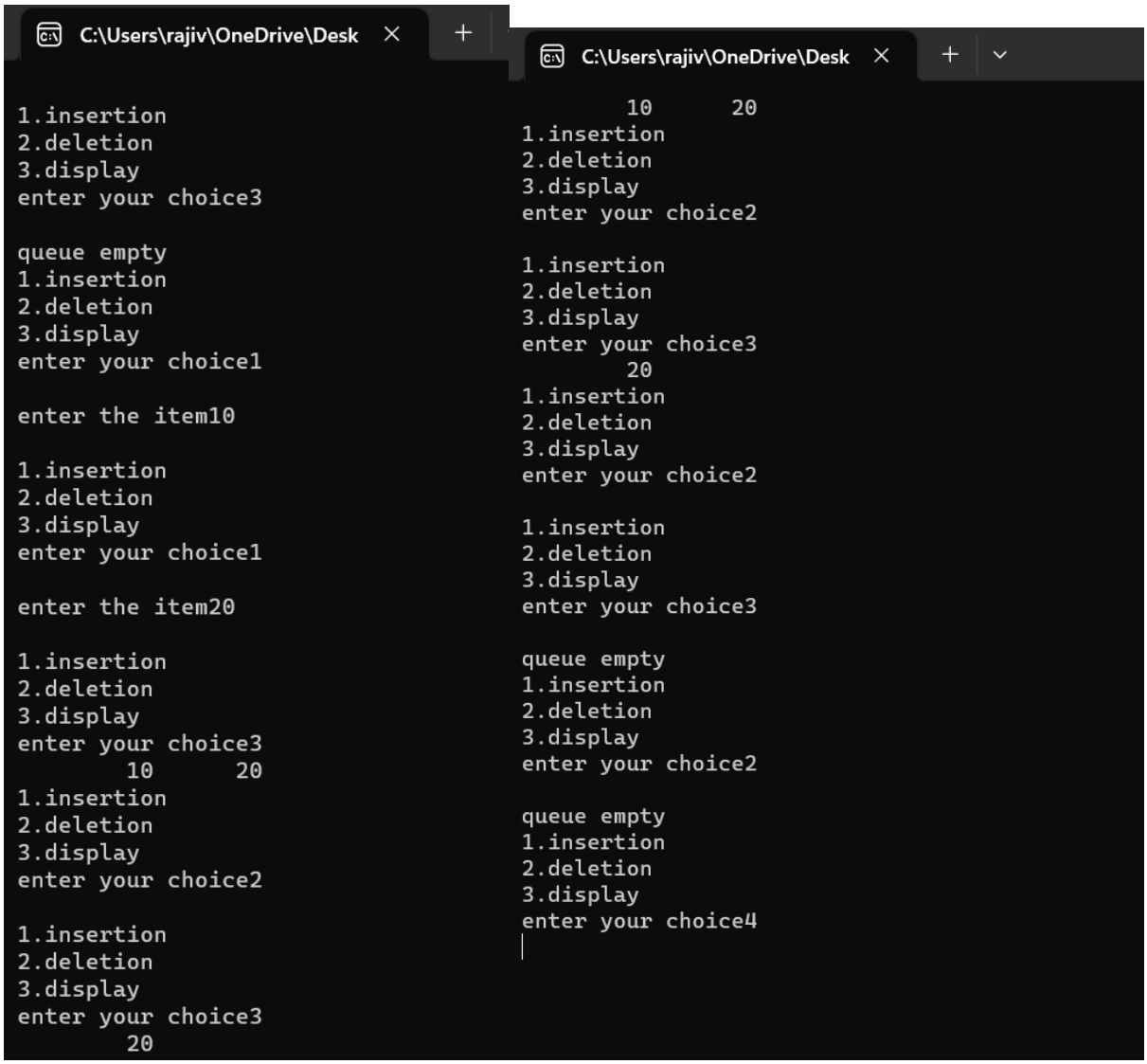
6. Write a C program to implement queue using linked list.

Source Code:

```
//queue using linked list
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
void insertion(void);
void deletion(void);
void display(void);
struct node
{
int data;
struct node *next;
};struct node
*nw,*front,*rear,*temp;
void main()
{
int choice;
nw=((struct node*)malloc(sizeof(struct node)));
nw->data=0;
nw->next=NULL;
front=nw;
rear=nw;
do
{
printf("\n1.insertion\n2.deletion\n3.display");
printf("\nenter your choice");
scanf("%d",&choice);
switch(choice)
{
case 1:
insertion();
break;
case 2:
deletion();
break;
case 3:
display();
break;
}}while(choice<4);
getch();
}
```

```
void insertion()
{
int item;
printf("\nenter the item");
scanf("%d",&item);
if(rear->data==0)
rear->data=item;
else
{
nw=((struct node*)malloc(sizeof(struct node)));
nw->data=item;
nw->next=NULL;
rear->next=nw;
rear=nw;
}}
void deletion()
{
if(rear->data==0)
printf("\nqueue empty");
else if(front!=rear)
front=front->next;
else
rear->data=0;
}
void display()
{
if(rear->data==0)
printf("\nqueue empty");
else
{
temp=front;
while(temp!=NULL)
{
printf("\t%d",temp->data);
temp=temp->next;
}}}
```

OUTPUT:


C:\Users\rajiv\OneDrive\Desk
1.insertion
2.deletion
3.display
enter your choice3
queue empty
1.insertion
2.deletion
3.display
enter your choice1
enter the item10
1.insertion
2.deletion
3.display
enter your choice1
enter the item20
1.insertion
2.deletion
3.display
enter your choice3
10 20
1.insertion
2.deletion
3.display
enter your choice2
1.insertion
2.deletion
3.display
enter your choice3
20
C:\Users\rajiv\OneDrive\Desk
10 20
1.insertion
2.deletion
3.display
enter your choice2
1.insertion
2.deletion
3.display
enter your choice3
20
1.insertion
2.deletion
3.display
enter your choice2
1.insertion
2.deletion
3.display
enter your choice3
queue empty
1.insertion
2.deletion
3.display
enter your choice2
queue empty
1.insertion
2.deletion
3.display
enter your choice4

7. Write a C program to implement polynomial addition using linked list.

SOURCE CODE:

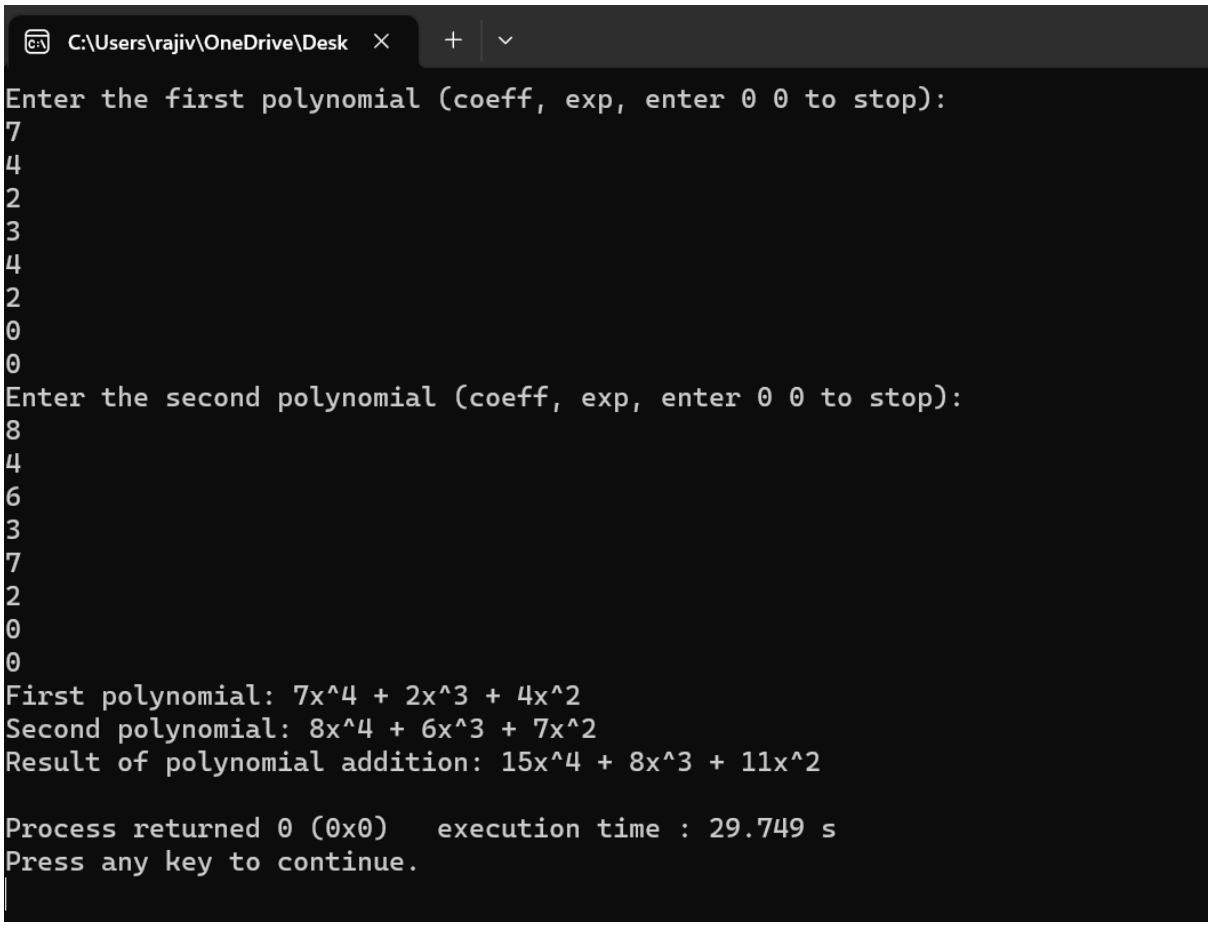
```
#include <stdio.h>
#include <stdlib.h>
struct Term {
    int coefficient;
    int exponent;
    struct Term* next;
};
struct Term* createTerm(int coeff, int exp) {
    struct Term* newTerm = (struct
Term*)malloc(sizeof(struct Term));
    if (newTerm == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newTerm->coefficient = coeff;
    newTerm->exponent = exp;
    newTerm->next = NULL;
    return newTerm;
}
void insertTerm(struct Term** poly, int coeff, int exp)
{
    struct Term* newTerm = createTerm(coeff, exp);
    if (*poly == NULL) {
        *poly = newTerm;
    } else {
        struct Term* current = *poly;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newTerm;
    }
}
void displayPoly(struct Term* poly) {
    if (poly == NULL) {
        printf("0\n");
        return;
    }
    while (poly != NULL) {
        printf("%dx^%d", poly->coefficient, poly-
>exponent);
        if (poly->next != NULL) {
```

```
            printf(" + ");
        }
        poly = poly->next;
    }
    printf("\n");
}
struct Term* addPolynomials(struct Term* poly1, struct
Term* poly2) {
    struct Term* result = NULL;
    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->exponent > poly2->exponent) {
            insertTerm(&result, poly1->coefficient,
poly1->exponent);
            poly1 = poly1->next;
        } else if (poly1->exponent < poly2->exponent) {
            insertTerm(&result, poly2->coefficient,
poly2->exponent);
            poly2 = poly2->next;
        } else {
            int sumCoeff = poly1->coefficient + poly2-
>coefficient;
            if (sumCoeff != 0) {
                insertTerm(&result, sumCoeff, poly1-
>exponent);
            }
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
    }
    while (poly1 != NULL) {
        insertTerm(&result, poly1->coefficient, poly1-
>exponent);
        poly1 = poly1->next;
    }
    while (poly2 != NULL) {
        insertTerm(&result, poly2->coefficient, poly2-
>exponent);
        poly2 = poly2->next;
    }
    return result;
}

int main() {
    struct Term* poly1 = NULL;
    struct Term* poly2 = NULL;
```

```
    struct Term* result = NULL;

    printf("Enter the first polynomial (coeff, exp,
enter 0 0 to stop):\n");
    int coeff, exp;
    while (1) {
        scanf("%d %d", &coeff, &exp);
        if (coeff == 0 && exp == 0) {
            break;
        }
        insertTerm(&poly1, coeff, exp);
    }

    printf("Enter the second polynomial (coeff, exp,
enter 0 0 to stop):\n");
    while (1) {
        scanf("%d %d", &coeff, &exp);
        if (coeff == 0 && exp == 0) {
            break;
        }
        insertTerm(&poly2, coeff, exp);
    }

    printf("First polynomial: ");
    displayPoly(poly1);
    printf("Second polynomial: ");
    displayPoly(poly2);

    result = addPolynomials(poly1, poly2);

    printf("Result of polynomial addition: ");
    displayPoly(result);

    while (poly1 != NULL) {
        struct Term* temp = poly1;
        poly1 = poly1->next;
        free(temp);
    }
    while (poly2 != NULL) {
        struct Term* temp = poly2;
        poly2 = poly2->next;
        free(temp);
    }
    while (result != NULL) {
        struct Term* temp = result;
```

```
        result = result->next;
        free(temp);
    }

    return 0;
}
```

Output:



8. Given two sorted lists L1 and L2 write a program to merge the two lists in sorted order.

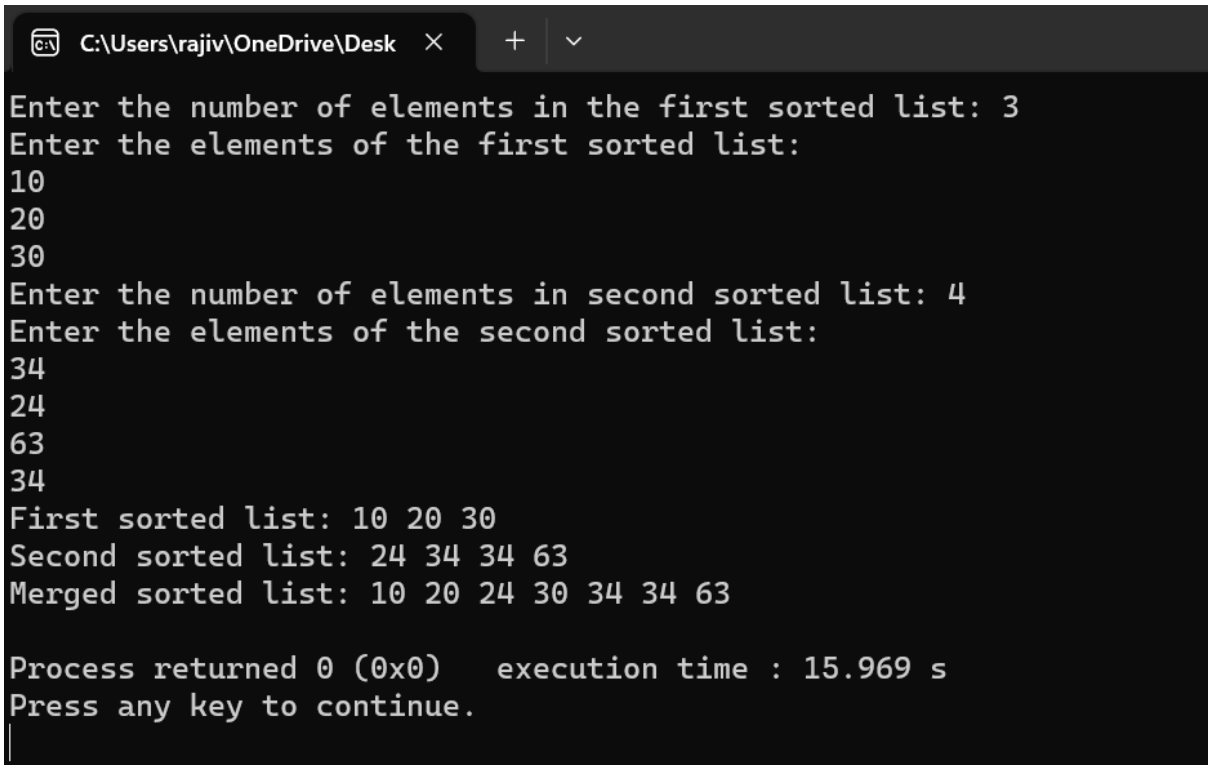
Source Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct Node* insertSorted(struct Node* head, int data)
{
    struct Node* newNode = createNode(data);
    if (head == NULL || data <= head->data) {
        newNode->next = head;
        return newNode;
    }
    struct Node* current = head;
    while (current->next != NULL && current->next->data
< data) {
        current = current->next;
    }
    newNode->next = current->next;
    current->next = newNode;
    return head;
}

struct Node* mergeSortedLists(struct Node* l1, struct
Node* l2) {
    struct Node* mergedList = NULL;
    while (l1 != NULL || l2 != NULL) {
```

```
        if (l1 != NULL && (l2 == NULL || l1->data <=
l2->data)) {
            mergedList = insertSorted(mergedList, l1-
>data);
            l1 = l1->next;
        } else if (l2 != NULL) {
            mergedList = insertSorted(mergedList, l2-
>data);
            l2 = l2->next;
        }
    }
    return mergedList;
}

void displaySorted(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    int n1, n2, num;

    printf("Enter the number of elements in the first
sorted list: ");
    scanf("%d", &n1);
    printf("Enter the elements of the first sorted
list:\n");
    for (int i = 0; i < n1; i++) {
        scanf("%d", &num);
        list1 = insertSorted(list1, num);
    }

    printf("Enter the number of elements in second
sorted list: ");
    scanf("%d", &n2);
    printf("Enter the elements of the second sorted
list:\n");
    for (int i = 0; i < n2; i++) {
```

```
        scanf("%d", &num);
        list2 = insertSorted(list2, num);
    }

    printf("First sorted list: ");
    displaySorted(list1);
    printf("Second sorted list: ");
    displaySorted(list2);

    struct Node* mergedList = mergeSortedLists(list1,
list2);

    printf("Merged sorted list: ");
    displaySorted(mergedList);

    while (list1 != NULL) {
        struct Node* temp = list1;
        list1 = list1->next;
        free(temp);
    }
    while (list2 != NULL) {
        struct Node* temp = list2;
        list2 = list2->next;
        free(temp);
    }
    while (mergedList != NULL) {
        struct Node* temp = mergedList;
        mergedList = mergedList->next;
        free(temp);
    }

    return 0;
}
```

OUTPUT:


C:\Users\rajiv\OneDrive\Desk
Enter the number of elements in the first sorted list: 3
Enter the elements of the first sorted list:
10
20
30
Enter the number of elements in second sorted list: 4
Enter the elements of the second sorted list:
34
24
63
34
First sorted list: 10 20 30
Second sorted list: 24 34 34 63
Merged sorted list: 10 20 24 30 34 34 63
Process returned 0 (0x0) execution time : 15.969 s
Press any key to continue.

9. Given two list L1 and L2 write a C program to find the intersection of two list.

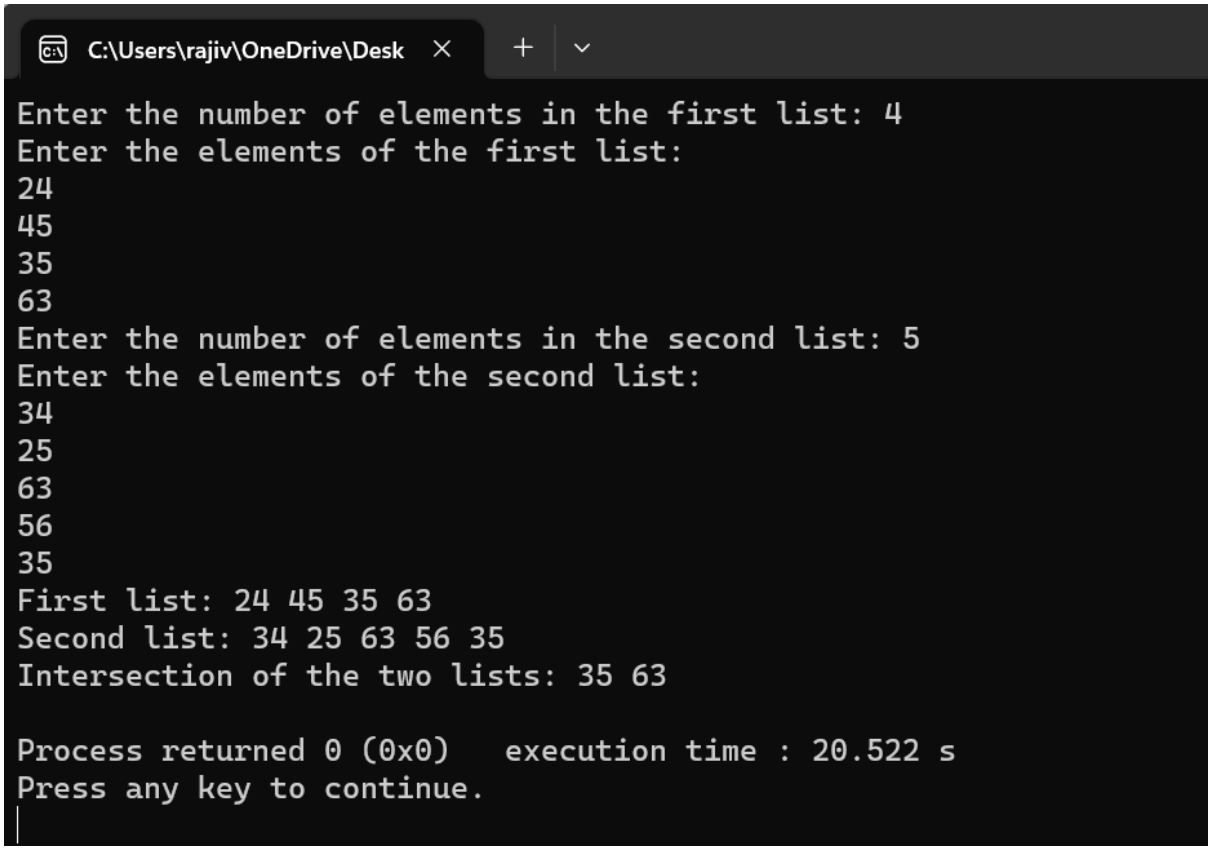
Source Code:

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
}

struct Node* findIntersection(struct Node* list1,
struct Node* list2) {
    struct Node* intersection = NULL;
    struct Node* current1 = list1;

    while (current1 != NULL) {
        struct Node* current2 = list2;
        while (current2 != NULL) {
```

```
            if (current1->data == current2->data) {
                insertEnd(&intersection, current1->data);
                break;
            }
            current2 = current2->next;
        }
        current1 = current1->next;
    }

    return intersection;
}

void displayList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    int n1, n2, num;

    printf("Enter the number of elements in the first list: ");
    scanf("%d", &n1);
    printf("Enter the elements of the first list:\n");
    for (int i = 0; i < n1; i++) {
        scanf("%d", &num);
        insertEnd(&list1, num);
    }

    printf("Enter the number of elements in the second list: ");
    scanf("%d", &n2);
    printf("Enter the elements of the second list:\n");
    for (int i = 0; i < n2; i++) {
        scanf("%d", &num);
        insertEnd(&list2, num);
    }
```

```
    printf("First list: ");
    displayList(list1);
    printf("Second list: ");
    displayList(list2);

    struct Node* intersection = findIntersection(list1,
list2);

    printf("Intersection of the two lists: ");
    displayList(intersection);

    while (list1 != NULL) {
        struct Node* temp = list1;
        list1 = list1->next;
        free(temp);
    }
    while (list2 != NULL) {
        struct Node* temp = list2;
        list2 = list2->next;
        free(temp);
    }
    while (intersection != NULL) {
        struct Node* temp = intersection;
        intersection = intersection->next;
        free(temp);
    }
  return 0;
}
```

OUTPUT:


C:\Users\rajiv\OneDrive\Desk
Enter the number of elements in the first list: 4
Enter the elements of the first list:
24
45
35
63
Enter the number of elements in the second list: 5
Enter the elements of the second list:
34
25
63
56
35
First list: 24 45 35 63
Second list: 34 25 63 56 35
Intersection of the two lists: 35 63
Process returned 0 (0x0) execution time : 20.522 s
Press any key to continue.

CHALLENGING EXPERIMENT

Assume FLAMES game that tests for relationship has to be implemented using a dynamic structure. The letters in the FLAMES stand for Friends, Love, Affection, Marriage, Enmity and Sister. Initially store the individual letters of the word 'flames' in the nodes of the dynamic structure. Given the count of the number of uncommon letters in the two names 'n', write a program to delete every nth node in it, till it is left with a single node. If the end of the dynamic structure is reached while counting, resume the counting from the beginning. Display the letter that still remains and the corresponding relationship
Eg., If Ajay and Jack are the two names, there are 4 uncommon letters in these. So delete 4th node in the first iteration and for the next iteration start counting from the node following the deleted node.

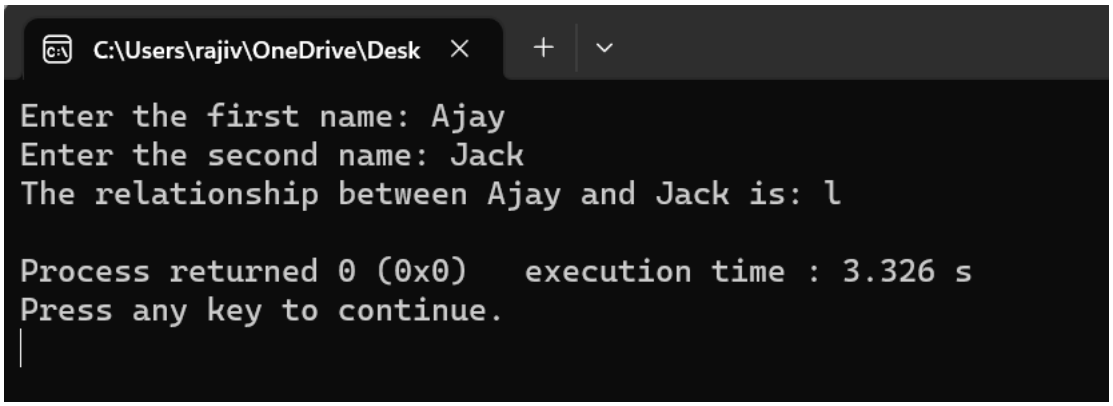
Source Code:

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

typedef struct Node {
  char data;
  struct Node* next;
} Node;

Node* create_node(char data) {
  Node* new_node = malloc(sizeof(Node));
  if (new_node == NULL) {
    printf("Memory allocation failed.\n");
    exit(1);
  }
  new_node->data = data;
  new_node->next = NULL;
  return new_node;
}

Node* insert_at_end(Node* head, char data) {
  Node* new_node = create_node(data);
  if (head == NULL) {
    head = new_node;
    new_node->next = new_node;
    return head;
  }
```

```
  Node* current = head;
  while (current->next != head) {
    current = current->next;
  }
  current->next = new_node;
  new_node->next = head;
  return head;
}

void delete_nth_node(Node* head, int n) {
  if (head == NULL) {
    printf("List is empty.\n");
    return;
  }

  Node* current = head;
  Node* previous = NULL;
  for (int i = 1; i <= n; i++) {
    previous = current;
    current = current->next;
  }

  if (previous == NULL) {
    // If the first node is being deleted
    if (current->next == current) {
      // Only one node is left
      free(current);
      head = NULL;
    } else {
      // Move head to the next node
      head = current->next;
    }
  } else {
    previous->next = current->next;
    free(current);
  }
}

char* flames(char* name1, char* name2) {
  Node* head = NULL;
  for (int i = 0; i < 6; i++) {
    head = insert_at_end(head, "flames"[i]);
  }
```

```
  int uncommon_letters = 0;
  for (int i = 0; name1[i] != '\0'; i++) {
    if (strchr(name2, name1[i]) == NULL) {
      uncommon_letters++;
    }
  }
  for (int i = 0; name2[i] != '\0'; i++) {
    if (strchr(name1, name2[i]) == NULL) {
      uncommon_letters++;
    }
  }


  Node* current = head;
  while (uncommon_letters > 0) {
    for (int i = 0; i < uncommon_letters; i++) {
      current = current->next;
    }
    delete_nth_node(head, uncommon_letters);
    uncommon_letters--;
  }

  return &current->data;
}

int main() {
  char name1[100];
  char name2[100];

  printf("Enter the first name: ");
  scanf("%s", name1);

  printf("Enter the second name: ");
  scanf("%s", name2);

  char* relationship = flames(name1, name2);

  printf("The relationship between %s and %s is: %s\n",
name1, name2, relationship);

  return 0;
}
```

Output:


C:\Users\rajiv\OneDrive\Desk
Enter the first name: Ajay
Enter the second name: Jack
The relationship between Ajay and Jack is: l
Process returned 0 (0x0)   execution time : 3.326 s
Press any key to continue.