Lab Assignment-5

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 24/08/21

QUES 1: [A] Write a menu driven program to perform the following operations in a single linked list by using suitable user defined functions for each case.

1. Write a program to search an element in a simple linked list, if found delete that node and insert that node at beginning. Otherwise display an appropriate message.

2. Write a program to find the middle node in a single linked list (without counting the number of nodes.)

3. Write a program to reverse the first m elements of a linked list of n nodes.

4. Write a program to check whether a given linked list is sorted or not.

5. Given a linked list which is sorted, write a program to insert an element into the linked list in sorted way.

6. Write a program to find the intersections elements of two linked list and store them in a third linked list.

7. Write a program to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list.

8. Write a program to check whether a singly linked list is a palindrome or not.

9. A linked list is said to contain a cycle if any node is visited more than once while traversing the list. Write a program to detect a cycle in a linked list.

10. Write a program to reverse only even position nodes in a singly linked list.

11. Write a program to swap kth node from beginning with kth node from end in a Linked List

12. Given a linked list, write a function to reverse every k nodes. (where k is an input to the function). If a linked list is given as 12->23->45->89->15->67->28->98->NULL and k=3 then output will be 45->23->12->67->15->89->98->28->NULL.

13. Given a singly linked list, rotate the linked list counter-clockwise by k nodes. Where k is a given positive integer. For example, if the given linked list is 10->20->30->40->50->60 and k is 4, the list should be modified to 50->60->10->20->30->40. Assume that k is smaller than the count of nodes in linked list.

SOLUTION:

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

typedef struct Node
{
    int data;
    struct Node *link;
} Node;

void menu();
void push(Node **, int);
void display(Node *);

void empty_the_list(Node **);
int search_and_replace_beg(Node **, int);
int middle_node_value(Node *);
void reverse_first_m_nodes(Node **, int);
int isSorted(Node *);
void sorted_push(Node **, int);
void make_unique(Node *);

Node *get_unique_list(Node *);
Node *make_intersection(Node *, Node *);

void even_first(Node **);
void reverse_the_nodes(Node **);
int isPalindrome(Node *);
int isCyclic(Node *);
void make_cycle(Node **, int);
void rev_even_position(Node *);
void swap_kth_pos(Node **, int);
void rev_every_k_nodes(Node **, int);
void rotate_cw_by_k(Node **, int);

int main()
{
    Node *list1 = NULL, *list2 = NULL, *list3 = NULL;
    int choice, pos, val;
    do
    {
        menu();
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter an element: ");
            scanf("%d", &val);
            push(&list1, val);
            break;
        case 2:
            printf("\n");
            display(list1);
            break;
        case 3:
            printf("\nEnter an element: ");
            scanf("%d", &val);
            push(&list2, val);
            break;
        case 4:
            printf("\n");
            display(list2);
            break;
        case 5:
            printf("\nList cleared!\n");
            empty_the_list(&list1);
            empty_the_list(&list2);
            empty_the_list(&list3);
            break;
        case 6:
            printf("\nEnter the value at node to connect last node to: ");
            scanf("%d", &val);
            make_cycle(&list1, val);
            break;
        case 7:
            printf("\nEnter the number to search for: ");
            scanf("%d", &val);
            if (!search_and_replace_beg(&list1, val))
            {
                printf("Number not found!\n");
                break;
            }
            printf("Now the list is: ");
            display(list1);
            break;
        case 8:
            printf("\nMiddle node has: ");
            printf("%d\n", middle_node_value(list1));
            break;
        case 9:
            printf("\nEnter the number of elements: ");
            scanf("%d", &pos);
            reverse_first_m_nodes(&list1, pos);
            printf("Now the list is: ");
            display(list1);
            break;
        case 10:
            printf(isSorted(list1) ? "\nTrue\n" : "\nFalse\n");
            break;
        case 11:
            printf("\nEnter element to insert: ");
            scanf("%d", &val);
            sorted_push(&list1, val);
            printf("Now the list is: ");
            display(list1);
            break;
        case 12:
            list3 = make_intersection(list1, list2);
            display(list3);
            break;
        case 13:
            printf("\nModified list: ");
            even_first(&list1);
            display(list1);
            break;
        case 14:
            printf("\nThe list is palindrome: ");
            printf(isPalindrome(list1) ? "true\n" : "false\n");
            break;
        case 15:
            printf("\nCycle is present: ");
            printf((isCyclic(list1) ? "true\n" : "false\n"));
            break;
        case 16:
            rev_even_position(list1);
            printf("\nThe modified list: ");
            display(list1);
            break;
        case 17:
            printf("\nEnter position to swap: ");
            scanf("%d", &pos);
            swap_kth_pos(&list1, pos);
            printf("The modified list: ");
            display(list1);
            break;
        case 18:
            printf("\nInsert value of k: ");
            scanf("%d", &pos);
            rev_every_k_nodes(&list1, pos);
            printf("The modified list: ");
            display(list1);
            break;
        case 19:
            printf("\nEnter tha value of k: ");
            scanf("%d", &pos);
            rotate_cw_by_k(&list1, pos);
            printf("The modified list: ");
            display(list1);
            break;
        default:
            printf("\nExiting...\n");
        }
        printf("---------------------------\n");
    } while (choice >= 1 && choice <= 19);
    return 0;
}
void menu()
{
    printf("Enter your choice: (all operations are on list 1, unless indicated otherwise)");
    printf("\n1) Create a new node \n2) Display the list \n");
    printf("3) Create a new node (list 2)\n4) Display the list (list 2)\n5) Drop all lists\n"
);
```

```c
    printf("6) Create a cycle\n7) Search and move to begining\n8) Middle node\n");
    printf("9) Reverse first m elements\n10) The list is sorted\n11) Insert in sorted order\n
");
    printf("12) Intersection of list 1 and 2\n13) Bring even elements at the begining\n");
    printf("14) Check if palindrome\n15) Check for cycles\n16) Even Position reverse\n");
    printf("17) Swap kth first and last node\n18) Reverse every k nodes\n");
    printf("19) Rotate the list in counter clockwise direction\n20) Exit\n->: ");
}
void push(Node **start, int n)
{
    /*Making the new node*/
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = n;
    temp->link = NULL;
    /*Connecting the new node*/
    if (!*start)
    {
        *start = temp;
        return;
    }
    Node *tempStart = *start;
    //Travel to last node
    while (tempStart->link)
        tempStart = tempStart->link;
    tempStart->link = temp; //  Insert after last node
}
void display(Node *start)
{
    if (isCyclic(start))
    {
        printf("List contains a cycle!\n");
        return;
    }
    while (start)
    {
        printf("%d->", start->data);
        start = start->link;
    }
    printf("null\n");
}
void empty_the_list(Node **start)
{
    while (*start)
    {
        Node *ptr = *start;
        *start = (*start)->link;
        free(ptr);
    }
}
int middle_node_value(Node *start)
```

```c
{
    if (!start)
        return -999;
    Node *slowPtr = start;
    Node *fastPtr = start;
    while (fastPtr && fastPtr->link)
    {
        fastPtr = fastPtr->link->link;
        slowPtr = slowPtr->link;
    }
    return slowPtr->data;
}
int search_and_replace_beg(Node **start, int num)
{
    if (!*start)
        return 0;
    Node *tempStart = *start;
    while (tempStart->link && tempStart->link->data != num)
    {
        tempStart = tempStart->link;
    }
    if (!tempStart->link)
        return 0;
    Node *ptr = tempStart->link;
    tempStart->link = tempStart->link->link;
    ptr->link = *start;
    *start = ptr;
    return 1;
}
void reverse_first_m_nodes(Node **start, int index)
{
    if (!*start)
        return;
    Node *temp = *start;              //current
    Node *temp1 = (*start)->link; //next
    Node *temp2;                          //previous
    while (temp1 && index - 1)
    {
        temp2 = temp1->link;
        temp1->link = temp;
        temp = temp1;
        temp1 = temp2;
        index--;
    }
    (*start)->link = temp1;
    *start = temp;
}
int isSorted(Node *start)
{
    if (!start)
```

```c
        return 1;
    while (start->link)
    {
        if (start->data > start->link->data)
            return 0;
        start = start->link;
    }
    return 1;
}
void sorted_push(Node **start, int num)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = num;
    temp->link = NULL;
    if (!*start)
    {
        *start = temp;
        return;
    }
    else if ((*start)->data > num)
    {
        temp->link = *start;
        *start = temp;
        return;
    }
    Node *tempStart = *start;
    while (tempStart->link && tempStart->link->data < num)
    {
        tempStart = tempStart->link;
    }
    temp->link = tempStart->link;
    tempStart->link = temp;
}
///////////////////////////////////////////////////////////////////////
void make_unique(Node *start)
{
    while (start)
    {
        Node *tempStart = start;
        while (tempStart->link)
        {
            if (tempStart->link->data == start->data)
            {
                Node *ptr = tempStart->link;
                tempStart->link = tempStart->link->link;
                free(ptr);
                continue;
            }
            tempStart = tempStart->link;
        }
        if (start)
            start = start->link;
    }
}
Node *get_unique_list(Node *start)
{
    Node *list = NULL;
    while (start)
    {
        Node *tempStart = start;
        while (tempStart->link)
        {
            if (tempStart->link->data == start->data)
            {
                push(&list, start->data);
            }
            tempStart = tempStart->link;
        }
        if (start)
            start = start->link;
    }
    make_unique(list);
    return list;
}
Node *make_intersection(Node *list1, Node *list2)
{
    if (!list1 || !list2)
        return NULL;
    Node *templ1 = list1;
    while (templ1->link)
    {
        templ1 = templ1->link;
    }
    templ1->link = list2;
    Node *list3 = get_unique_list(list1);
    templ1->link = NULL;
    return list3;
}
////////////////////////////////////////////////////////////////////
void even_first(Node **start)
{
    Node *prev = NULL;
    Node *current = *start;
    while (current)
    {
        if (current->data % 2 == 0 && prev)
        {
            prev->link = current->link;
            current->link = *start;
            *start = current;
            current = prev;
        }
        prev = current;
        current = current->link;
    }
}
void reverse_the_nodes(Node **start)
{
    if (!*start)
        return;
    Node *temp = *start;              //current
    Node *temp1 = (*start)->link; //next
    Node *temp2;                          //previous
    while (temp1 != NULL)
    {
        temp2 = temp1->link;
        temp1->link = temp;
        temp = temp1;
        temp1 = temp2;
    }
    (*start)->link = NULL;
    *start = temp;
}
int isPalindrome(Node *start)
{
    if (!start)
        return 1;
    Node *slowPtr = NULL;
    Node *fastPtr = start;
    while (fastPtr && fastPtr->link)
    {
        //for first run slow ptr is updated as the head
        slowPtr = (!slowPtr) ? fastPtr : slowPtr->link;
        fastPtr = fastPtr->link->link;
    }
    Node *firstTale = slowPtr;
    Node *midnode = firstTale; //taking the tale of the first list as mid node
    if (!slowPtr)
        return 1;      //if the list contains only one element return true
    else if (!fastPtr) //if the list length is even
    {
        //Make the fast pointer point to the head of secoond list(even)
        fastPtr = slowPtr->link;
    }
    else if (!fastPtr->link) //if the list length is odd
    {
        //Make the fast pointer point to the head of the second list(odd)
        fastPtr = slowPtr->link->link;
        midnode = slowPtr->link; //the node after the tale node is the mid node
    }
    slowPtr->link = NULL;        //Split the list in the middle
    reverse_the_nodes(&fastPtr); //reverse the second list
    Node *midHead = fastPtr;
    int flag = 1;
    while (fastPtr) //compare the 2 lists until the head is null
    {
        if (start->data != fastPtr->data)
        {
            flag = 0;
            break;
        }
        start = start->link;
        fastPtr = fastPtr->link;
    }
    //reverse the second list again and append list 1 to list 2
    reverse_the_nodes(&midHead);
    firstTale->link = midnode;
    midnode->link = midHead;
    return flag;
}
int isCyclic(Node *start)
{
    Node *slowPtr = start;
    Node *fastPtr = start;
    while ((fastPtr && fastPtr->link))
    {
        fastPtr = fastPtr->link->link;
        slowPtr = slowPtr->link;
        if (slowPtr == fastPtr)
            return 1;
    }
    return 0;
}
void make_cycle(Node **start, int val)
{
    if (!*start)
        return;
    Node *tale = *start;
    Node *cycle_head = NULL;
    if ((*start)->data == val) //If first node is the only node
    {
        cycle_head = *start;
    }
    while (tale->link)
    {
        if (tale->data == val)
            cycle_head = tale;
        tale = tale->link;
    }
    tale->link = cycle_head;
```

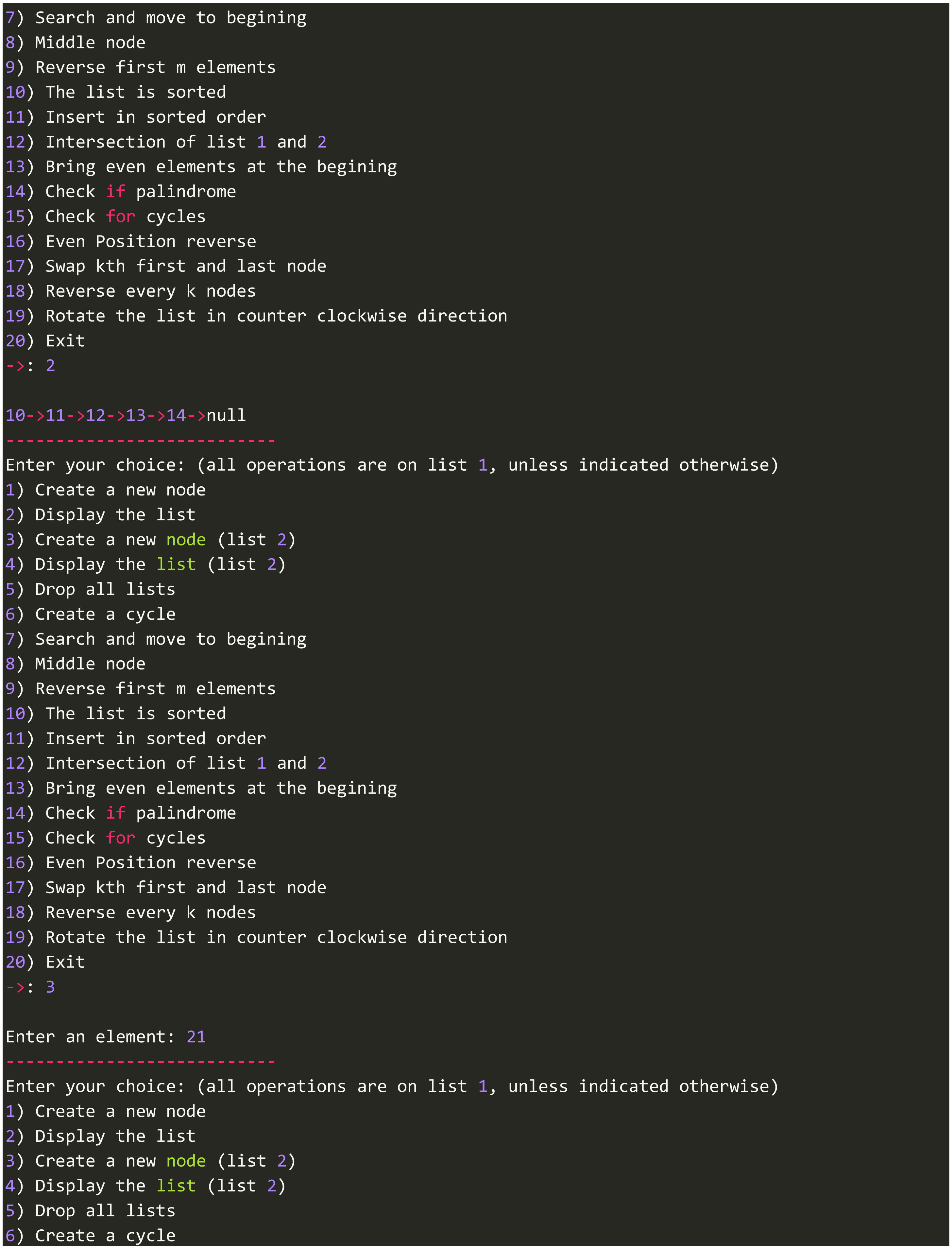
```c
}
void rev_even_position(Node *start)
{
    Node *even = NULL;
    Node *odd = start;
    Node *temp = NULL;
    while (odd && odd->link)
    {
        temp = odd->link;
        odd->link = temp->link;
        temp->link = even;
        even = temp;
        odd = odd->link;
    }
    odd = start;
    while (odd && odd->link)
    {
        temp = even;
        even = even->link;
        temp->link = odd->link;
        odd->link = temp;
        odd = temp->link;
    }
    if (odd)
        odd->link = even;
}
void swap_kth_pos(Node **start, int k)
{
    int length = 0;
    Node *tempStart = *start;
    while (tempStart)
    {
        tempStart = tempStart->link;
        length++;
    }
    if (k > length || k <= 0)
        return;
    else if (k * 2 - 1 == length)
        return;
    tempStart = *start;
    for (int i = 1; i < k - 1; i++)
    {
        tempStart = tempStart->link;
    }
    Node *tempEnd = *start;
    for (int i = 1; i < length - k; i++)
    {
        tempEnd = tempEnd->link;
    }
    Node *nowStart = tempStart->link;
    Node *nowEnd = tempEnd->link;
    if (k == 1)
        nowStart = tempStart;
    else
        tempStart->link = nowEnd;
    tempEnd->link = nowStart;
    Node *temp = nowStart->link; //Swap the links of the kth nodes
    nowStart->link = nowEnd->link;
    nowEnd->link = temp;
    if (k == 1)
        *start = nowEnd;
    else if (k == length)
        *start = nowStart;
}
void rev_every_k_nodes(Node **start, int k)
{
    if (!*start || k == 1)
        return;
    Node *holder = (Node *)malloc(sizeof(Node));
    holder->link = *start; //Making a place holder node before head
    Node *prev = holder;
    Node *current = prev->link; //current is next to previous
    Node *next = current->link; //next is next to current
    while (next)
    {
        current = prev->link;
        next = current->link;
        for (int i = 1; i < k; i++)
        {
            if (!next)
                break;
            current->link = next->link;
            next->link = prev->link;
            prev->link = next;
            next = current->link;
        }
        prev = current;
    }
    *start = holder->link; //Update the head node
    free(holder);
}
void rotate_cw_by_k(Node **start, int k)
{
    if (!*start)
        return;
    /*Goes to and splits the list upto where to rotate.*/
    Node *tempStart = *start;
    while (k - 1 && tempStart->link)
    {
        tempStart = tempStart->link;
        k--;
    }
    Node *secondaryHead = tempStart->link;
    tempStart->link = NULL;
    tempStart = secondaryHead;
    if (!secondaryHead)
        return;
    /*Appends the first list to the second list*/
    while (secondaryHead->link)
    {
        secondaryHead = secondaryHead->link;
    }
    secondaryHead->link = *start;
    *start = tempStart;
}
```

OUTPUT:

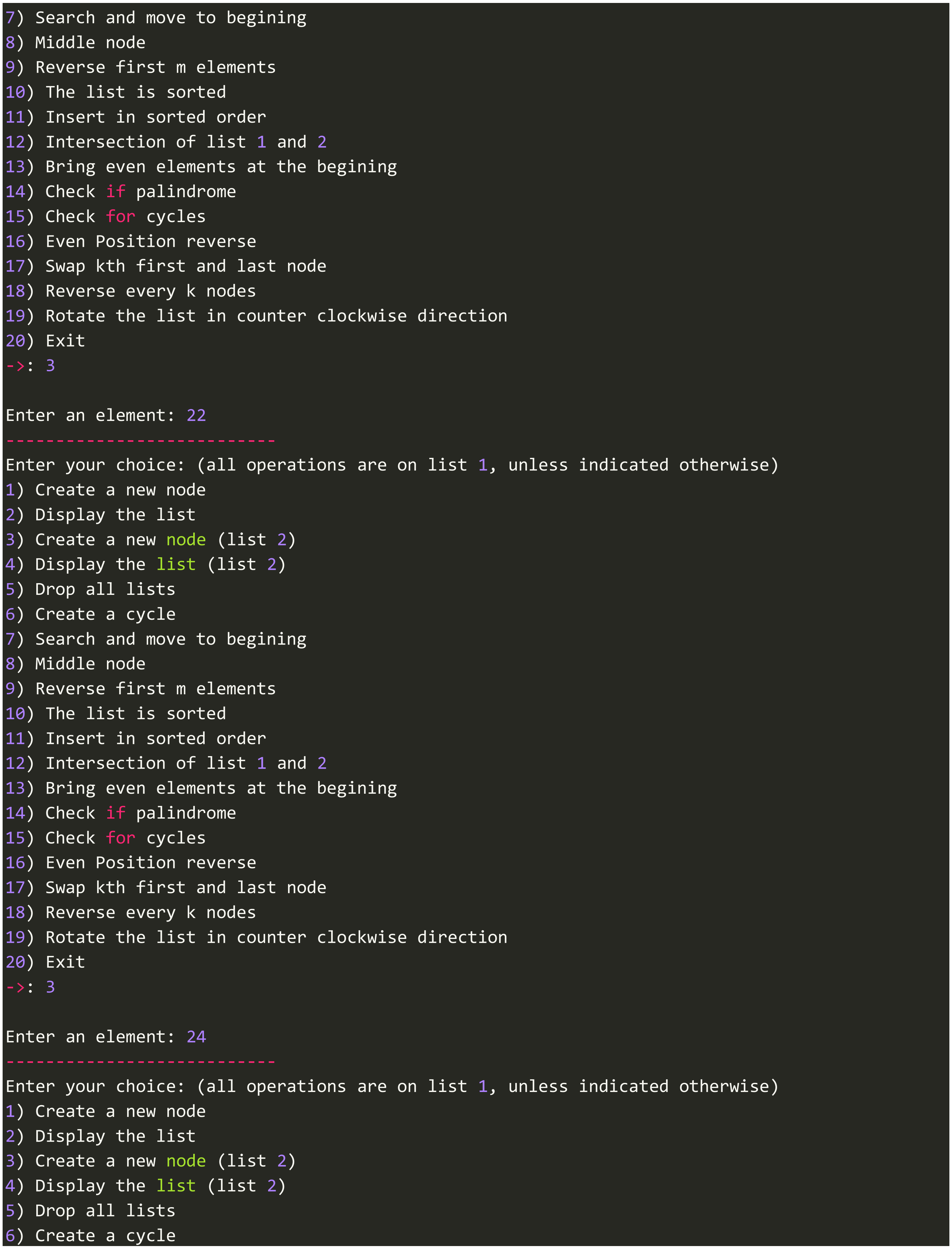
```
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 1

Enter an element: 10
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
```

```
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 1

Enter an element: 11
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 1

Enter an element: 12
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
```

```
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 1

Enter an element: 13
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 1

Enter an element: 14
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
```



```
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 2

10->11->12->13->14->null
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 3

Enter an element: 21
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
```



```
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 3

Enter an element: 22
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 3

Enter an element: 24
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
```

```
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 3

Enter an element: 25
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 4

21->22->24->25->null
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
```

```
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
14) Check if palindrome
15) Check for cycles
16) Even Position reverse
17) Swap kth first and last node
18) Reverse every k nodes
19) Rotate the list in counter clockwise direction
20) Exit
->: 6

Enter the value at node to connect last node to: 10
---------------------------
Enter your choice: (all operations are on list 1, unless indicated otherwise)
1) Create a new node
2) Display the list
3) Create a new node (list 2)
4) Display the list (list 2)
5) Drop all lists
6) Create a cycle
7) Search and move to begining
8) Middle node
9) Reverse first m elements
10) The list is sorted
11) Insert in sorted order
12) Intersection of list 1 and 2
13) Bring even elements at the begining
se direction
20) Exit
->: 20

Exiting...
---------------------------
```

QUES 2: [B] Write a program to merge two sorted linked list.

SOLUTION:

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

typedef struct Node
{
    int data;
    struct Node *link;
```

```c
} Node;

void push(Node **, int);
void disp(Node *);
struct Node *mergeTwoLists(Node *, Node *);

int main()
{
    Node *l1 = NULL;
    Node *l2 = NULL;
    char chos;

    printf("Insert elements in list 1:\n");
    do
    {
        int data;
        printf("Enter a element: ");
        scanf("%d", &data);
        push(&l1, data);
        printf("Insert another element?: ");
        scanf(" %c", &chos);
    } while (chos == 'y' || chos == 'Y');

    printf("\nInsert elements in list 2:\n");
    do
    {
        int data;
        printf("Enter a element: ");
        scanf("%d", &data);
        push(&l2, data);
        printf("Insert another element?: ");
        scanf(" %c", &chos);
    } while (chos == 'y' || chos == 'Y');

    Node *l3 = mergeTwoLists(l1, l2);
    printf("\nMerged list:\n");
    disp(l3);
    return 0;
}

void push(Node **head, int n)
{
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    temp->data = n;
    temp->link = NULL;
    if (*head == NULL)
    {
        *head = temp;
        return;
    }
    Node *ptr = *head;
    while (ptr->link != NULL)
    {
        ptr = ptr->link;
    }
    ptr->link = temp;
}

void disp(Node *head)
{
    while (head != NULL)
    {
        printf("%i->", head->data);
        head = head->link;
    }
    printf("\b\b \n");
}

Node *mergeTwoLists(Node *l1, Node *l2)
{
    if (!l1)
        return l2;
    else if (!l2)
        return l1;

    Node *l3 = NULL;
    Node *copyOfl3;

    while (l2 && l1)
    {
        Node *temp = (Node *)malloc(sizeof(Node));
        temp->link = NULL;

        if (l1->data <= l2->data)
        {
            temp->data = l1->data;
            l1 = l1->link;
        }
        else
        {
            temp->data = l2->data;
            l2 = l2->link;
        }

        if (!l3)
            copyOfl3 = l3 = temp;
        copyOfl3->link = temp;
        copyOfl3 = copyOfl3->link;

        if (!l1)
```

```
            copyOfl3->link = l2;
        else if (!l2)
            copyOfl3->link = l1;
    }
    return l3;
}
```

OUTPUT:

```
Insert elements in list 1:
Enter a element: 10
Insert another element?: y
Enter a element: 12
Insert another element?: y
Enter a element: 14
Insert another element?: y
Enter a element: 15
Insert another element?: n

Insert elements in list 2:
Enter a element: 7
Insert another element?: y
Enter a element: 8
Insert another element?: y
Enter a element: 13
Insert another element?: y
Enter a element: 16
Insert another element?: n

Merged list:
7->8->10->12->13->14->15->16
```

QUES 3: [C] Write a program to represent a polynomial using linked list. Write a function to add two polynomials.

```
SOLUTION: #include <stdio.h>

#include <stdlib.h>

typedef struct Node
{
    int coff;
    int expo;
    struct Node *link;
} Node;

void push(Node **, int, int);
void display(Node *);
Node *add_polynomial(Node *, Node *);
```

```c
int main()
{
    Node *l1 = NULL;
    Node *l2 = NULL;
    char chos;
    printf("Enter first polynomial:\n");
    do
    {
        int expo, coff;
        printf("Enter coffecient and exponent: ");
        scanf("%d%d", &coff, &expo);
        push(&l1, coff, expo);
        printf("Insert another element?: ");
        scanf(" %c", &chos);
    } while (chos == 'y' || chos == 'Y');
    printf("\nEnter second polynomial:\n");
    do
    {
        int expo, coff;
        printf("Enter coffecient and exponent: ");
        scanf("%d%d", &coff, &expo);
        push(&l2, coff, expo);
        printf("Insert another element?: ");
        scanf(" %c", &chos);
    } while (chos == 'y' || chos == 'Y');
    Node *l3 = add_polynomial(l1, l2);
    printf("\nThe resultant polynomial is:\n");
    display(l3);
    return 0;
}

void push(Node **start, int coff, int expo)
{
    /*Making the new node*/
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->coff = coff;
    temp->expo = expo;
    temp->link = NULL;
    /*Connecting the new node*/
    if (!*start)
    {
        *start = temp;
        return;
    }
    Node *tempStart = *start;
    //Travel to last node
    while (tempStart->link)
        tempStart = tempStart->link;
    tempStart->link = temp;
    //Insert after last node
```

```c
}

void display(Node *start)
{
    if (!start)
        printf("0");
    while (start)
    {
        if (start->coff > 0)
            printf("+");
        printf("%dx^%d", start->coff, start->expo);
        start = start->link;
    }
    printf("\n");
}

Node *add_polynomial(Node *list1, Node *list2)
{
    Node *addList = NULL;
    while (list1 && list2)
    {
        if (list1->expo > list2->expo)
        {
            push(&addList, list1->coff, list1->expo);
            list1 = list1->link;
            continue;
        }
        else if (list1->expo < list2->expo)
        {
            push(&addList, list2->coff, list2->expo);
            list2 = list2->link;
            continue;
        }
        if (list1->coff + list2->coff)
        {
            push(&addList, list2->coff + list1->coff, list2->expo);
            list2 = list2->link;
            list1 = list1->link;
        }
    }
    while (list1)
    {
        push(&addList, list1->coff, list1->expo);
        list1 = list1->link;
    }
    while (list2)
    {
        push(&addList, list2->coff, list2->expo);
        list2 = list2->link;
    }
    return addList;
}
```

OUTPUT:

```
Enter first polynomial:
Enter coffecient and exponent: 3 1
Insert another element?: y
Enter coffecient and exponent: 2 2
Insert another element?: y
Enter coffecient and exponent: 7 3
Insert another element?: n

Enter second polynomial:
Enter coffecient and exponent: 2 1
Insert another element?: y
Enter coffecient and exponent: 7 2
Insert another element?: y
Enter coffecient and exponent: 4 3
Insert another element?: n

The resultant polynomial is:
+5x^1+9x^2+11x^3
```

QUES 4: [D] Write a program to represent a sparse matrix in three tuple format using an array and perform addition.

SOLUTION:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50

int **create_matrix(int, int);
void free_matrix(int **, int);
void convert_to_tuple(int **, int[][3], int, int);
void display_tuple(int[][3]);
void insert(int arr[][3], int row, int a, int b, int c);
void add_tuple(int[][3], int[][3], int[][3]);

int main()
{
    int tupleA[MAX][3], tupleB[MAX][3], tupleC[MAX][3];
    int row, col;
    printf("Enter rows and columns: ");
    scanf("%d%d", &row, &col);
    printf("Enter a sparse matrix (no.1):\n");
    int **arr1 = create_matrix(row, col);
    printf("Enter a sparse matrix (no.2):\n");
    int **arr2 = create_matrix(row, col);
```

```c
    convert_to_tuple(arr1, tupleA, row, col);
    convert_to_tuple(arr2, tupleB, row, col);
    add_tuple(tupleA, tupleB, tupleC);
    printf("The resultant tuple:\n");
    display_tuple(tupleC);
    free_matrix(arr1, row);
    free_matrix(arr2, row);
    return 0;
}

int **create_matrix(int row, int col)
{
    int **arr = (int **)malloc(row * sizeof(int *));
    for (int i = 0; i < row; i++)
        arr[i] = (int *)malloc(col * sizeof(int));
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            scanf("%d", &arr[i][j]);
    return arr;
}

void free_matrix(int **arr, int row)
{
    for (int i = 0; i < row; i++)
        free(arr[i]);
    free(arr);
}

void convert_to_tuple(int **arr, int tuple[][3], int row, int col)
{
    int count = 0, i, j;
    tuple[0][0] = row;
    tuple[0][1] = col;
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            if (arr[i][j] != 0)
            {
                tuple[++count][0] = i;
                tuple[count][1] = j;
                tuple[count][2] = arr[i][j];
            }
        }
    }
    tuple[0][2] = count;
}

void display_tuple(int arr[][3])
{
    int i, j;
    for (i = 0; i <= arr[0][2]; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%d ", arr[i][j]);
        printf("\n");
    }
}

void insert(int arr[][3], int row, int a, int b, int c)
{
    arr[row][0] = a;
    arr[row][1] = b;
    arr[row][2] = c;
}

void add_tuple(int arr1[][3], int arr2[][3], int arr3[][3])
{
    if (arr1[0][0] != arr2[0][0] || arr1[0][1] != arr2[0][1])
    {
        printf("The operattion is not possible!\n");
        return;
    }
    int rowA = 1, rowB = 1;
    int row = 1;
    arr3[0][0] = arr1[0][0];
    arr3[0][1] = arr1[0][1];
    while (rowA <= arr1[0][2] && rowB <= arr2[0][2])
    {
        //if(arr1>arr2) result=arr2: If row is smaller, or row is same but column is smaller
        if (arr1[rowA][0] > arr2[rowB][0] || (arr1[rowA][0] == arr2[rowB][0] &&
                                              arr1[rowA][1] > arr2[rowB][1]))
        {
            insert(arr3, row++, arr2[rowB][0], arr2[rowB][1], arr2[rowB][2]);
            rowB++;
            continue;
        }
        //if(arr2>arr1) result=arr1: If row is smaller, or row is same but column is smaller
        else if (arr1[rowA][0] < arr2[rowB][0] || (arr1[rowA][0] == arr2[rowB][0] &&
                                                   arr1[rowA][1] < arr2[rowB][1]))
        {
            insert(arr3, row++, arr1[rowA][0], arr1[rowA][1], arr1[rowA][2]);
            rowA++;
            continue;
        }
        //if(arr1==arr2) result=arr2+arr1: If row and column both are same
        insert(arr3, row++, arr2[rowB][0], arr2[rowB][1], arr2[rowB][2] + arr1[rowA][2]);
        rowB++;
        rowA++;
    }
    for (; rowA <= arr1[0][2]; rowA++)
        insert(arr3, row++, arr1[rowA][0], arr1[rowA][1], arr1[rowA][2]);
    for (; rowB <= arr2[0][2]; rowB++)
        insert(arr3, row++, arr2[rowB][0], arr2[rowB][1], arr2[rowB][2]);
    arr3[0][2] = row - 1;
}
```

OUTPUT:

```
Enter rows and columns: 3 3
Enter a sparse matrix (no.1):
1 0 0
0 2 0
0 0 0
Enter a sparse matrix (no.2):
1 0 0
2 3 0
0 1 0
The resultant tuple:
3 3 4
0 0 2
1 0 2
1 1 5
2 1 1
```