**1. Write a C program to find Odd or Even number from a given set of numbers**

Code:

```
#include <stdio.h>

int main()

{

int num;

printf("Enter the number of elements: ");

scanf("%d", &num);

printf("Enter the elements: ");

for (int i = 0; i < num; i++) {

 int element;

 scanf("%d", & element);

  if (element % 2 == 0) {

   printf("%d is even\n", element);

  } else {

    printf("%d is odd\n", element);

       }

    }

    return 0;

}
```

Output:**Enter the number of elements: 4**

**Enter the elements: 3**

**3 is odd**

**4**

**4 is even**

**7**

**7 is odd**

**2**

**2 is even**

**2.. Write a C program to find Factorial of a given number using Recursion**

Code:

```
#include <stdio.h>
long int fact(int x) {
    if (x >= 1)
        return x * fact(x - 1);
    else
        return 1;
}
int main() {
    int x;
    printf("Enter a number to find factorial: ");
    scanf("%d", &x);
    printf("The factorial of %d = %ld", x, fact(x));
```

```
    return 0;
}
```

Output:**Enter a number to find factorial: 5**

**The factorial of 5 = 120**

### 3. Write a C program to perform Matrix Multiplication

Code:

```
#include <stdio.h>
int main() {
 nt m, n, p, q, i, j, k;
   printf("Enter the number of rows and columns of first matrix: ");
   scanf("%d %d", &m, &n);
   printf("Enter the number of rows and columns of second matrix: ");
   scanf("%d %d", &p, &q);
  if (n != p) {
      printf("Matrices with entered orders can't be multiplied with each other.\n");
   } else {
      int first[m][n], second[p][q], multiply[m][q];
printf("Enter elements of first matrix:\n");
      for (i = 0; i < m; i++) {
         for (j = 0; j < n; j++) {
            scanf("%d", & first[i][j]);
```

```
        }
    }
printf("Enter elements of second matrix:\n");
    for (i = 0; i < p; i++) {
        for (j = 0; j < q; j++) {
            scanf("%d", & second[i][j]);
        }
    }

    for (i = 0; i < m; i++) {
        for (j = 0; j < q; j++) {
            multiply[i][j] = 0;
            for (k = 0; k < n; k++) {
                multiply[i][j] += first[i][k] * second[k][j];
            }
        }
    }

    printf("Product of the matrices:\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < q; j++) {
            printf("%d\t", multiply[i][j]);
        }
```

```
            printf("\n");
        }
    }
 return 0;
}
```

Output:**Enter the number of rows and columns of first matrix: 2 2**

**Enter the number of rows and columns of second matrix: 2 2**

**Enter elements of first matrix:**

**1 2**

**3 4**

**Enter elements of second matrix:**

**5 6**

**7 8**

**Product of the matrices:**

**19 22**

**43 50**

**4. Write a program that uses functions to perform the following operations on singly linked list i) Creation ii) Insertion iii) Deletion iv) Traversal.**

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));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertNode(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

void deleteNode(struct Node** head, int key) {
    struct Node *temp = *head, *prev;

    if (temp != NULL && temp->data == key) {
        *head = temp->next;
```

```
        free(temp);
        return;
    }

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

    if (temp == NULL) return;

    prev->next = temp->next;
    free(temp);
}

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

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

    insertNode(&head, 3);
    insertNode(&head, 5);
    insertNode(&head, 7);

    printf("Linked list after creation and insertions:\n");
    traverseList(head);

    deleteNode(&head, 5);
    printf("Linked list after deletion of 5:\n");
    traverseList(head);

    return 0;
}
```

Output: **Linked list after creation and insertions:**

**7 -> 5 -> 3 -> NULL**

**Linked list after deletion of 5:**

**7 -> 3 -> NULL**

**5. Write a program that uses functions to perform the following operations on doubly linked lists i) Creation ii) Insertion iii) Deletion iv) Traversal.**

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
void insertNode(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
}
```

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

```
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* head = NULL;
     insertNode(&head, 3);
    insertNode(&head, 5);
    insertNode(&head, 7);
     printf("Doubly linked list after creation and insertions:\n");
    traverseList(head);
   deleteNode(&head, 5);
    printf("Doubly linked list after deletion of 5:\n");
    traverseList(head);
   return 0;
}
```

Output:**Doubly linked list after creation and insertions:**

**7 -> 5 -> 3 -> NULL**

**Doubly linked list after deletion of 5:**

**7 -> 3 -> NULL**

**6. Write a program that uses functions to perform the following operations on circular linked List i) Creation ii) Insertion iii) Deletion iv) Traversal.**

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

```
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
    }
}
void deleteNode(struct Node** head, int key) {
    if (*head == NULL) return;
struct Node *current = *head, *prev;
 while (current->data != key) {
        if (current->next == *head) {
            printf("%d not found in the list\n", key);
            return;
        }
        prev = current;
        current = current->next;
    }
  if (current->next == *head && prev == NULL) {
        *head = NULL;
        free(current);
        return;
    }
 if (current == *head) {
```

```
        prev = *head;
        while (prev->next != *head) {
            prev = prev->next;
        }
        *head = current->next;
        prev->next = *head;
    } else if (current->next == *head) {
        prev->next = *head;
    } else {
        prev->next = current->next;
    }
    free(current);
}
void traverseList(struct Node* head) {
    struct Node* temp = head;
    if (head != NULL) {
        do {
            printf("%d -> ", temp->data);
            temp = temp->next;
        } while (temp != head);
        printf(" (Head)\n");
    }
}
```

```
int main() {
    struct Node* head = NULL;
 insertNode(&head, 3);
    insertNode(&head, 5);
    insertNode(&head, 7);
  printf("Circular linked list after creation and insertions:\n");
    traverseList(head);
   deleteNode(&head, 5);
    printf("Circular linked list after deletion of 5:\n");
    traverseList(head);
return 0;
}
```

Output: **Circular linked list after creation and insertions:**

**3 -> 5 -> 7 ->  (Head)**

**Circular linked list after deletion of 5:**

**3 -> 7 ->  (Head)**