In [8]:

```
# Defining matrices

def null_matx(r,c): # making a null matrix
    # r is for number of rows and c for columns
    X = []
    for i in range(r):
        Y = []
        for j in range(c):
            Y.append(0)
        X.append(Y)
    return X

print(null_matx(2,4)) # test run

# direct input as a list

def make_matx(val,r,c): # val is for the values imported from the txt file
    Nu = null_matx(r,c)
    for i in range(r):
        for j in range(c):                      # the string of numbers is marked as
            Nu[i][j] = float(val[c*i + j])      # running through all columns of the
    return Nu
# coversion example: 103945 to [[1,0],[3,9],[4,5]] (preserving the order, the desire
```

```
[[0, 0, 0, 0], [0, 0, 0, 0]]
```

In [68]:

```
# taking input from a txt file
def gen_matx(file,r,c):
    p = open(file,"r")
    f = p.read()            # extracting the data
    z = make_matx(f,r,c)    # representing the matrix as a list
    return z

# generating raw output (as nested list)

A = gen_matx("matA.txt",3,3)
print(A)
```

```
[[2.0, 3.0, 4.0], [9.0, 8.0, 7.0], [1.0, 5.0, 9.0]]
```

In [69]:

```
def disp_matx(lst): # lst is the matrix input as a list
    print("The matrix representation is: ")
    for i in range(len(lst)):
        t = 0                   # using t as a counter
        while t < len(lst[0]): # running through each column per specified row
            dum = lst[i][t]    # define a dummy variable
            print(" " + str(dum)+ " ",end=" ")
            t += 1
        print("\n")
    print("---")


disp_matx(A)
```

```
The matrix representation is:
 2.0   3.0   4.0

 9.0   8.0   7.0

 1.0   5.0   9.0

---
```

In [72]:

```
B = gen_matx("matB.txt",3,3)
C = gen_matx("matC.txt",1,3)
D = gen_matx("matD.txt",3,1)

disp_matx(B)
disp_matx(C)
disp_matx(D)
```

```
The matrix representation is:
 1.0   0.0   0.0

 0.0   2.0   0.0

 0.0   0.0   3.0

---
The matrix representation is:
 2.0   7.0   4.0

---
The matrix representation is:
 1.0

 0.0

 1.0

---
```

In [75]:

```
# adding matrices

def add_matx(a,b):
    if len(a) == len(b) and len(a[0]) == len(b[0]): # check step for invalid operati
        X = null_matx(len(a),len(a[0]))
        for i in range(len(a)):
            for j in range(len(a[0])):
                X[i][j] = a[i][j] + b[i][j]        # individual addition
        return X
    else:
        print("Please input matrices of same order.")

anb = add_matx(A,B)
disp_matx(anb)
```

```
The matrix representation is:
 3.0   3.0   4.0

 9.0   10.0   7.0

 1.0   5.0   12.0

---
```

In [76]:

```
# multiplying matrices

def prod_matx(a,b):
    X = null_matx(len(a),len(b[0])) # null matrix of order of desired output matrix
    if len(a[0]) == len(b):         # check step
        for i in range(len(a)):              # keeping row of 1st matrix fixed
            for j in range(len(b[0])):       # keeping column of 2nd matrix fixed
                for k in range(len(a[0])):   # len(a[0]) == len(b) so we run through
                    X[i][j] += a[i][k]*b[k][j]  # multiplication step
```

```
        return X
    else:
        print("Multiplication is undefined for these two matrices.")

CD = prod_matx(C,D)
AB = prod_matx(A,B)
CA = prod_matx(C,A)
BD = prod_matx(B,D)

disp_matx(CD)
disp_matx(AB)
disp_matx(CA)
disp_matx(BD)
```

```
The matrix representation is:
 6.0

---
The matrix representation is:
 2.0   6.0   12.0

 9.0   16.0   21.0

 1.0   10.0   27.0

---
The matrix representation is:
 71.0   82.0   93.0

---
The matrix representation is:
 1.0

 0.0

 3.0

---
```