****

**Q1) Write a program to multiply two given matrices.**

**/\***

**Write a program to multiply two given matrices.**

**CODE BY 2020ITB028\_RUPAYAN\_THAKUR**

**\*/**

**#include <stdio.h>**

**int main()**

**{**

**int m, n, p, q, sum = 0;**

**int first[10][10], second[10][10], multiply[10][10];**

**printf("Enter the number of rows and columns of first matrix\n");**

**scanf("%d%d", &m, &n);**

**printf("Enter the elements of first matrix\n");**

**for (  int i = 0 ; i < m ; i++ )**

**for ( int j = 0 ; j < n ; j++ )**

**scanf("%d", &first[i][j]);**

**printf("Enter the number of rows and columns of second matrix\n");**

**scanf("%d%d", &p, &q);**

**if ( n != p )**

**printf("Matrices with entered orders can't be multiplied with each other.\n");**

**else**

**{**

**printf("Enter the elements of second matrix\n");**

**for ( int i = 0 ; i < p ; i++ )**

**for ( int j = 0 ; j < q ; j++ )**

**scanf("%d", &second[i][j]);**

**for ( int i = 0 ; i < m ; i++ )**

**{**

**for ( int j = 0 ; j < q ; j++ )**

**{**

**for ( int k = 0 ; k < p ; k++ )**

**{**

**sum = sum + first[i][k]\*second[k][j];**

**}**

**multiply[i][j] = sum;**

**sum = 0;**

**}**

**}**

**printf("Product of entered matrices: \n");**

**for ( int i = 0 ; i < m ; i++ )**

**{**

**for ( int j = 0 ; j < q ; j++ )**

**printf("%d\t", multiply[i][j]);**

**printf("\n");**

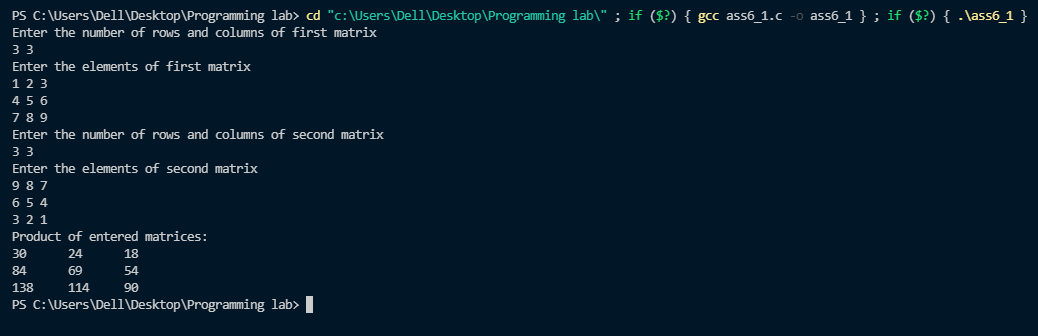
**}**

**}**

**return 0;**

**}**

**OUTPUT:**

****

**Q2) Write another program to multiply a chain of matrices. Use a function to multiply two matrices which is to be called from the main.**

**/\***

**Write another program to multiply a chain of matrices.**

**Use a function to multiply two matrices which is to be called from the main.**

**CODE BY 2020ITB028\_RUPAYAN\_THAKUR**

**\*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct matrix**

**{**

**int r;**

**int c;**

**int\*\* values;**

**};**

**void createMatrix(struct matrix\* m, int r, int c)**

**{**

**m->values= malloc(r\*sizeof(int\*));**

**for(int i= 0; i<r; i++)**

**m->values[i]= malloc(c\*sizeof(int));**

**m->r= r;**

**m->c= c;**

**}**

**void fillValues(struct matrix\* m)**

**{**

**printf("Enter values for matrix:\n");**

**for(int i= 0; i<m->r; i++)**

**for(int j= 0; j<m->c; j++)**

**scanf("%d", &m->values[i][j]);**

**}**

**struct matrix matMult(struct matrix\* m1, struct matrix\* m2)**

**{**

**struct matrix result;**

**createMatrix(&result, m1->r, m2->c);**

**for(int i= 0; i<result.r; i++)**

**for(int j= 0; j<result.c; j++)**

**{**

**result.values[i][j]= 0;**

**for(int x= 0; x<m1->c; x++)**

**result.values[i][j]+= m1->values[i][x]\*m2->values[x][j];**

**}**

**return result;**

**}**

**void display(struct matrix\* m)**

**{**

**printf("\nMatrix:\n");**

**for(int i= 0; i<m->r; i++)**

**{**

**for(int j= 0; j<m->c; j++)**

**printf("%d ", m->values[i][j]);**

**printf("\n");**

**}**

**}**

**void main()**

**{**

**int n;**

**printf("Enter number of matrices: ");**

**scanf("%d", &n);**

**struct matrix m1, m2, result;**

**int r,c;**

**printf("Enter number of rows in Matrix 1: ");**

**scanf("%d", &r);**

**printf("Enter number of columns in Matrix 1: ");**

**scanf("%d", &c);**

**createMatrix(&m1, r, c);**

**fillValues(&m1);**

**for(int i= 2; i<=n; i++)**

**{**

**r= m1.c;**

**printf("Enter number of columns in Matrix %d: ", i);**

**scanf("%d", &c);**

**createMatrix(&m2, r, c);**

**fillValues(&m2);**

**result= matMult(&m1, &m2);**

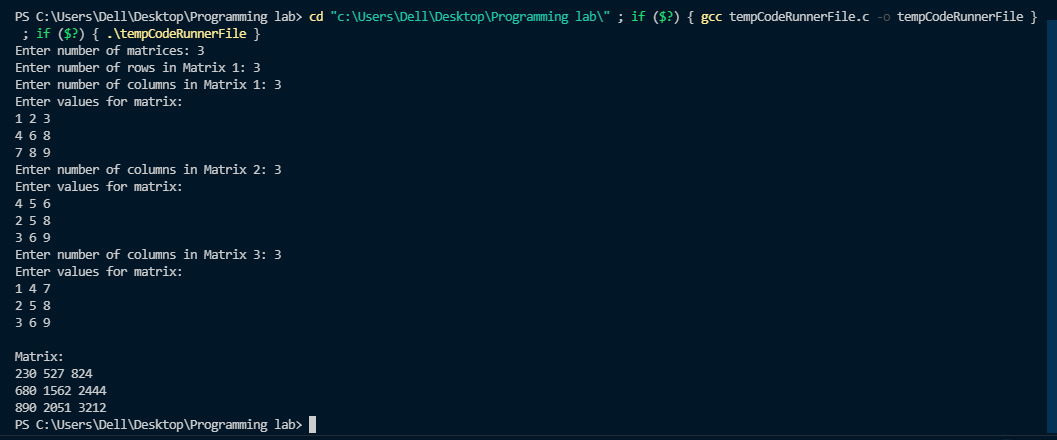
**m1= result;**

**}**

**display(&result);**

**}**

**OUTPUT:**

****

* **By using different orders it can be seen that execution time for the first order takes much less time as compared to the second order as in the first order includes less number of calculations. So, basically the lesser the order number more is the efficiency of the program and vice versa.**