MATRIX MULTIPLICATION

Dr. Rahul Das Gupta

Inner Product

Inner Product of a Row-vector and Column Vector is defined as follows:

$$\left[a_{i,0}, a_{i,1}, a_{i,2}, \ldots, a_{i,n-1}\right]\begin{bmatrix} b_{0,k} \\ b_{1,k} \\ b_{2,k} \\ \ldots \\ \ldots \\ \ldots \\ b_{n-1,k} \end{bmatrix} = \sum_{j=0}^{n-1} a_{i,j} * b_{j,k}$$

Basic Rules for Matrix Multiplication

#1: Necessary Condition: Two matrices can only be multiplied if

No. of columns of the 1st Matrix = No. of rows of the 2nd Matrix

#2: Matrix Multiplication is impossible if

No. of columns of the 1st Matrix != No. of rows of the 2nd Matrix

#3: (i, k) th element of the Product Matrix

= Inner Product between i th Row Vector of the 1st Matrix and k th Column Vector of the 2nd Matrix

$$[A]_{m\times n} * [B]_{n\times p} = [C]_{m\times p}$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \ldots & a_{0,\mathrm{n}-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \ldots & a_{1,\mathrm{n}-1} \\ \ldots & \ldots & \ldots & & \ldots \\ a_{\mathrm{i},0} & a_{\mathrm{i},1} & a_{\mathrm{i},2} & \ldots & a_{\mathrm{i},\mathrm{n}-1} \\ \ldots & \ldots & \ldots & & \ldots \\ a_{\mathrm{m}-1,0} & a_{\mathrm{m}-1,1} & a_{\mathrm{m}-1,2} & \ldots & a_{\mathrm{m}-1,\mathrm{n}-1} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & \ldots & b_{0,k} & \ldots & b_{0,p-1} \\ b_{1,0} & b_{1,1} & b_{1,2} & \ldots & b_{1,k} & \ldots & b_{1,p-1} \\ b_{2,0} & b_{2,1} & b_{2,2} & \ldots & b_{2,k} & \ldots & b_{2,p-1} \\ \ldots & \ldots & \ldots & & \ldots & & \ldots \\ b_{\mathrm{n}-1,1} & b_{\mathrm{n}-1,2} & b_{\mathrm{n}-1,3} & \ldots & b_{\mathrm{n}-1,k} & \ldots & b_{\mathrm{n}-1,p-1} \end{bmatrix}$$

$$= \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & \ldots & c_{0,k} & \ldots & c_{0,p-1} \\ c_{1,0} & c_{1,1} & c_{1,2} & \ldots & c_{1,k} & \ldots & c_{1,p-1} \\ \ldots & \ldots & \ldots & & \ldots & & \ldots \\ c_{\mathrm{i},0} & c_{\mathrm{i},1} & c_{\mathrm{i},2} & \ldots & c_{\mathrm{i},k} & \ldots & c_{\mathrm{i},p-1} \\ \ldots & \ldots & \ldots & & \ldots & & \ldots \\ c_{\mathrm{m}-1,0} & c_{\mathrm{m}-1,1} & c_{\mathrm{m}-1,2} & \ldots & c_{\mathrm{m}-1,k} & \ldots & c_{\mathrm{m}-1,p-1} \end{bmatrix}$$

where

$$c_{i,k} = \left[a_{i,0}, a_{i,1}, a_{i,2}, \ldots, a_{i,n-1}\right] \begin{bmatrix} b_{0,k} \\ b_{1,k} \\ b_{2,k} \\ \ldots \\ \ldots \\ \ldots \\ b_{n-1,k} \end{bmatrix} = \sum_{j=0}^{n-1} a_{i,j} * b_{j,k}$$

$$[A]_{m\times n} * [B]_{n\times p} = [C]_{m\times p}$$

k th Column Vector

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & & \cdots \\ a_{i,0} & a_{i,1} & a_{i,2} & \cdots & a_{i,n-1} \\ \cdots & \cdots & \cdots & & \cdots \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & \cdots & b_{0,k} & \cdots & b_{0,p-1} \\ b_{1,0} & b_{1,1} & b_{1,2} & \cdots & b_{1,k} & \cdots & b_{1,p-1} \\ b_{2,0} & b_{2,1} & b_{2,2} & \cdots & b_{2,k} & \cdots & b_{2,p-1} \\ \cdots & \cdots & \cdots & & \cdots & & \cdots \\ b_{n-1,1} & b_{n-1,2} & b_{n-1,3} & \cdots & b_{n-1,k} & \cdots & b_{n-1,p-1} \end{bmatrix}$$

i th Row Vector

i th Row

$$= \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & \cdots & c_{0,k} & \cdots & c_{0,p-1} \\ c_{1,0} & c_{1,1} & c_{1,2} & \cdots & c_{1,k} & \cdots & c_{1,p-1} \\ \cdots & \cdots & \cdots & & \cdots & & \cdots \\ c_{i,0} & c_{i,1} & c_{i,2} & \cdots & c_{i,k} & \cdots & c_{i,p-1} \\ \cdots & \cdots & \cdots & & \cdots & & \cdots \\ c_{m-1,0} & c_{m-1,1} & c_{m-1,2} & \cdots & c_{m-1,k} & \cdots & c_{m-1,p-1} \end{bmatrix}$$

where $$\boxed{c_{i,k} = \sum_{j=0}^{n-1} a_{i,j} * b_{j,k}}$$

k th Column

Algorithm for Matrix Multiplication:

$$[A]_{m\times n} * [B]_{n\times p} = [C]_{m\times p}$$

$$c_{i,k} = \left[a_{i,0}, a_{i,1}, a_{i,2}, \ldots, a_{i,n-1} \right] \begin{bmatrix} b_{0,k} \\ b_{1,k} \\ b_{2,k} \\ \ldots \\ \ldots \\ \ldots \\ b_{n-1,k} \end{bmatrix} = \sum_{j=0}^{n-1} a_{i,j} * b_{j,k}$$

$$\textit{for all } i = 0,1,2,\ldots,m-1$$

$$\textit{for all } k = 0,1,2,\ldots,p-1$$

$for\,(i = 0;\ i < m;\ i++)$

$\quad for(k = 0; k < p; k++)$

$\quad \{$

$\qquad c[i][k] = 0.0;$

$\qquad for(\mathrm{j} = 0;\ j < n;\ j++)$

$\qquad\quad \mathrm{c}[i][k] += a[i][j] * b[j][k];$

$\quad \}$

Elements of the Product Matrix

$$c_{i,k} = \boxed{\sum_{j=0}^{n-1}} a_{i,j} * b_{j,k} \quad \boxed{\textit{for all } i = 0,1,2,\ldots,m-1} \quad \boxed{\textit{for all } k = 0,1,2,\ldots,p-1}$$

$\boxed{for\,(i = 0;\ i < m;\ i++)}$

$\quad \boxed{for(k = 0; k < p; k++)}$

$\quad \{$

Don't forget to set c[i][k]=0.0

$\qquad c[i][k] = 0.0;$

$\qquad \boxed{for(\mathrm{j} = 0;\ j < n;\ j++)}$

$\qquad\quad \mathrm{c}[i][k] += a[i][j] * b[j][k];$

$\quad \}$

Computational Complexity Analysis

$$
\begin{aligned}
&for\,(i=0;\ i<m;\ i++)\\
&\quad for(k=0; k<p; k++)\\
&\quad \{\\
&\qquad c[i][k]=0.0;\\
&\qquad for(\mathrm{j}=0;\ j<n;\ j++)\\
&\qquad\quad \mathrm{c}[i][k]+=a[i][j]*b[j][k];\\
&\quad \}
\end{aligned}
$$

$Total\ \ no.\ \text{of}\ elementary\ \ multiplication\ \ operations = m\times p\times n$

$Total\ \ no.\ \text{of}\ elementary\ \ addition\ \ operations = m\times p\times n$

Structure of a Matrix

```
#define MAX 10
struct matrix
{
    int n_row, n_col;
    int  A[MAX][MAX];
};
struct matrix M1, M2, M3;

struct matrix *p;

/*p is a pointer to a struct matrix (i.e., Data type of p is struct matrix *)
*/
```

The name of new composite data type 'struct matrix' can alternatively be declared in the following way:

```
# define MAX 10

typedef struct
{
  int  n_row, n_col;
  int  A[MAX][MAX];
}MATRIX;
```

MATRIX M1, M2, M3;

MATRIX *p;

/*p is a pointer to a MATRIX (i.e., Data type of p is MATRIX *) */

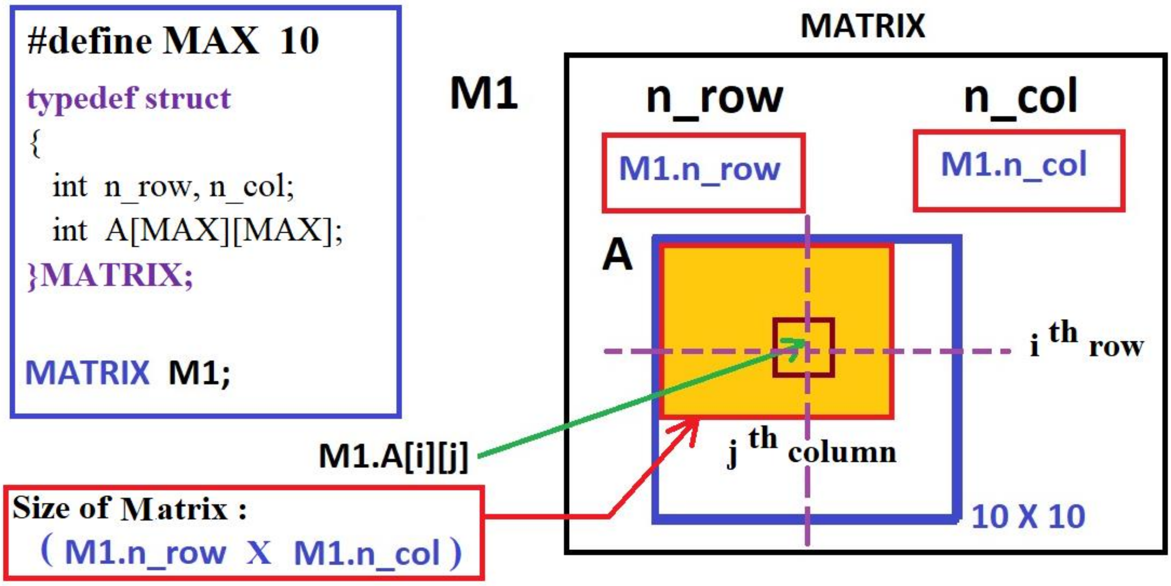
Accessing Elements of the Structure MATRIX:

 # define MAX 10

```
typedef struct
{
   int  n_row, n_col;
   int  A[MAX][MAX];
}MATRIX;
MATRIX M1, M2, M3;
```

MATRIX *p;

/*p is a pointer to a MATRIX (i.e., Data type of p is MATRIX *) */



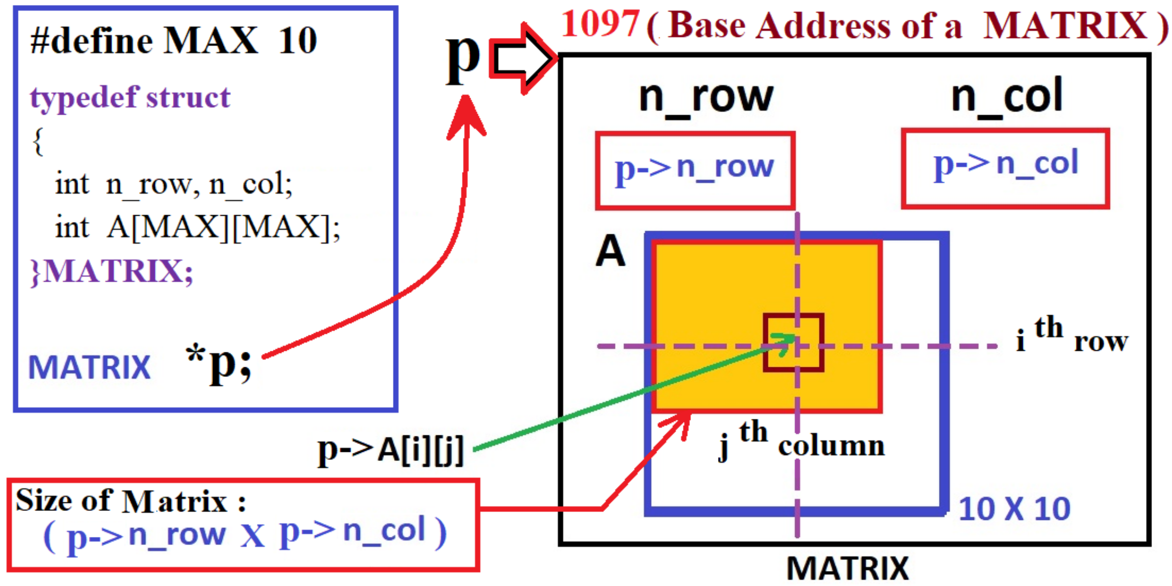
Case 1 (Dot) : Member Accessing Operator . (for non pointer variable)

Here,

M1.n_row denotes No. of rows of the MATRIX M1 (a non pointer variable)

M1.n_col denotes No. of columns of the MATRIX M1 (a non pointer variable)

M1.A[i][j] denotes the element corresponding to i th row and j th column of the MATRIX M1 (a non-pointer variable).



Case 2 (Arrow) : Member Accessing Operator -> (for pointer variable)

Here, p is a pointer to a structure MATRIX.

Data type of p : MATRIX *

p->n_row denotes No. of rows of the MATRIX pointed by a pointer p (a pointer variable)

p->n_col denotes No. of columns of the MATRIX pointed by a pointer p (a pointer variable)

p->A[i][j] denotes the element corresponding to i th row and j th column of the MATRIX pointed by a pointer p (a pointer variable).

C Function for Matrix Multiplication

1st Approach [*Address of the Matrix* (holding the product) passed as the input argument]:

Here, M1 and M2 are two matrices to be multiplied.

We are not going to modify M1 and M2.

[Important Concept of C Language to be remembered:**

In C Language, whenever a *variable is modified using a C function*, the *address of the variable to be modified* must be passed as an *input argument to the C function*.]

Here, the *address of the matrix* &R *holding the product* of M1 and M2 must be passed as *an input argument to the function* multiplication.

Look at the following statement in the ***main*** function:

multiplication (M1, M2, &R);

In following function definition ***pointer variable*** p (Data type: MATRIX *) will be replaced by **&R (*address of the matrix holding the product of two matrices*).**

void multiplication (MATRIX M1, MATRIX M2, MATRIX *p)

```
/*=====================================================
        M1: 1st Matrix,
        M2: 2nd Matrix
        p: pointer to (address of) the product Matrix
=====================================================*/
{
   int  i,j,k;
   if(M1.n_col!=M2.n_row)
   {
       printf("\n Matrix multiplication is impossible…");
       return;
    }
    p->n_row=M1.n_row;
    p->n_col=M2.n_col;
    for(i=0;i<p->n_row; i++)
         for(k=0;k<p->n_col; k++)
             {
                 p->A[i][k]=0;
                 for(j=0; j<M1.n_col; j++)
                           p->A[i][k]+=M1.A[i][j]*M2.A[j][k];
```

```
        }
}
```

2nd Approach **[A MATRIX type structure is returned]:**

```
MATRIX  multiply(MATRIX  M1,MATRIX  M2)
{
   MATRIX T;
   int  i, j, k;
   if(M1.n_col!=M2.n_row)
   {
       printf("\n Matrix multiplication is impossible…");
       exit(1);
    }
    T.n_row=M1.n_row;
    T.n_col=M2.n_col;
    for(i=0;i<T.n_row; i++)
         for(k=0;k<T.n_col; k++)
            {
                T.A[i][k]=0;
                for(j=0; j<M1.n_col; j++)
                        T.A[i][k]+=M1.A[i][j]*M2.A[j][k];
            }
     return T;
}
```

A Sample C Program for Matrix Multiplication

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10

typedef struct
{
  int  n_row, n_col;
```

```
  int  A[MAX][MAX];
}MATRIX;


void input (MATRIX *);
void display (MATRIX);
void multiplication (MATRIX, MATRIX, MATRIX *);
MATRIX  multiply(MATRIX ,MATRIX );

void main( )
{
   MATRIX  M1, M2, R, T;
   clrscr( );
   input(&M1);
   input(&M2);
   multiplication(M1,M2, &R);
   display(R);
    T=multiply(M1,M2);
    display(T);
    getchar( );
}

void input (MATRIX *p)
{
   int  i, j;
   printf("\n Enter the no. of rows:");
   fflush(stdin);
   scanf("%d", &p->n_row);
   printf("\n Enter the no. of columns:");
   fflush(stdin);
   scanf("%d", &p->n_col);

   for(i=0;i<p->n_row; i++)
        for(j=0;j<p->n_col; j++)
          {
              printf("\n Enter Element [%d][%d]:",i+1,j+1);
              fflush(stdin);
              scanf("%d", &p->A[i][j]);
```

```
        }
}


void display (MATRIX  M)
{
  int  i, j;
  for(i=0;i<M.n_row; i++)
  {
     printf("\n");
     for(j=0;j<M.n_col; j++)
          printf( "%2d",M.A[i][j]);
   }
}

void multiplication (MATRIX M1, MATRIX M2, MATRIX *p)
{
  int  i,j,k;
  if(M1.n_col!=M2.n_row)
  {
     printf("\n Matrix multiplication is impossible…");
     return;
   }
   p->n_row=M1.n_row;
   p->n_col=M2.n_col;
   for(i=0;i<p->n_row; i++)
       for(k=0;k<p->n_col; k++)
          {
             p->A[i][k]=0;
             for(j=0; j<M1.n_col; j++)
                   p->A[i][k]+=M1.A[i][j]*M2.A[j][k];
          }
}
```

```
MATRIX  multiply(MATRIX  M1,MATRIX  M2)
{
   MATRIX T;
   int  i, j, k;
   if(M1.n_col!=M2.n_row)
   {
       printf("\n Matrix multiplication is impossible…");
       exit(1);
    }
    T.n_row=M1.n_row;
    T.n_col=M2.n_col;
    for(i=0;i<T.n_row; i++)
         for(k=0;k<T.n_col; k++)
             {
                 T.A[i][k]=0;
                 for(j=0; j<M1.n_col; j++)
                           T.A[i][k]+=M1.A[i][j]*M2.A[j][k];
             }
     return T;
}
```