

UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE INGENIERÍAS CAMPUS IRAPUATO-SALAMANCA

LIC. EN ING. EN SISTEMAS COMPUTACIONALES

ALGORITMOS Y ESTRUCTURAS DE DATOS

PROFESOR: DR. CARLOS HUGO GARCÍA CAPULÍN

NO. DE TAREA: 07

NOMBRE DE LA TAREA:

ESTRUCTURA MATRIZ

ESTUDIANTE:

MANRÍQUEZ COBIÁN ROGELIO

FECHA DE ENTREGA:

22 DE SEPTIEMBRE DEL 2020



Problema

Se implementará un programa en el cual tendrás que capturar dos matrices con ayuda de una estructura de datos que contenga los datos a capturar de renglones y columnas, estas matrices a capturar tendrás que manejarlas con el uso de funciones, apuntadores y memoria dinámica.

Cuando ya tengas las matrices capturadas, se harán las dos operaciones aritméticas con ellas,

- Suma de Matrices

Recordar cómo se hace la suma de dos matrices:

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 3 & 0 & 0 \\ 5 & 1 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

$$A + B = \begin{pmatrix} 2+1 & 0+0 & 1+1 \\ 3+1 & 0+2 & 0+1 \\ 5+1 & 1+1 & 1+0 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 2 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{pmatrix}$$

- Multiplicación de Matrices

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 \\ 0 & 5 & 2 \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix}$$

3 x 2 2 x 3 3 x 3

Si se pueden multiplicar

Solución Implementada:

```

Clase13.c  Clase10.c
1  #include <stdio.h>
2  #include <stdlib.h> //Librería para usar los apartados de memoria dinamica
3
4
5  //Declaramos nuestra estructura con el nombre Matriz
6  typedef struct{
7      //Datos tipo Flotantes
8
9      float *data;
10     //Apuntador en cual se almacenará la dirección de un arreglo de datos bidimensionales
11
12     //Declarar variables para renglones y columnas
13     unsigned int nRen; //No. renglones
14     unsigned int nCol; //No. columnas
15
16 }Matriz;
17
18 //Prototipos de Funciones
19 Matriz* rcCrearMatriz (unsigned int nRen, unsigned int nCol);
20 //Reserva la memoria necesaria para almacenar una matriz
21
22 void rcCapturaMatriz(Matriz *p);
23 //Captura los datos en la matriz apuntada por "p"
24
25 void rcMostarMatriz(Matriz *p);
26 //Manda a mostrar la matriz apuntada por "p"
27
28 void rcEliminarMatriz (Matriz *p);
29 //Liberar la memoria reservada
30
31 Matriz* sumaMatrices (Matriz *a, Matriz *b);
32 //Suma A + B
33
34 Matriz* multiplicaMatrices (Matriz *a, Matriz *b);
35 //Multiplica A*B
36
37 int main (void){
38     Matriz *a, *b, *c, *d;
39
40     unsigned int ren, col;
41
42     printf ("Numero de renglones: ");
43     scanf ("%u", &ren); fflush (stdin);
44
45     printf ("Numero de columnas: ");
46     scanf ("%u", &col); fflush (stdin);
47
48     printf ("\nMATRIZ [A]\n");
49     a = rcCrearMatriz(ren,col);
50     rcCapturaMatriz(a);
51
52     printf("\nLa matriz [A] capturada es: \n");
53     rcMostarMatriz(a);
54
55     printf ("\n\nMATRIZ [B]\n");
56     b = rcCrearMatriz(ren,col);
57     rcCapturaMatriz(b);
58
59     printf("\nLa matriz [B] capturada es: \n");
60     rcMostarMatriz(b);
61
62     //Suma
63     c = sumaMatrices(a,b);
64     printf("\n\nLa matriz [A]+[B] es igual: \n");
65     rcMostarMatriz(c);
66
67     //Multiplicar
68     d = multiplicaMatrices(a,b);
69     printf("\n\nLa matriz [A]*[B] es igual: \n");
70     rcMostarMatriz(d);
71
72

```

```

73 //Eliminar Matriz
74 rcEliminarMatriz(a);
75 rcEliminarMatriz(b);
76 rcEliminarMatriz(c);
77 rcEliminarMatriz(d);
78
79 printf ("\n\nPresione <ENTER> para salir");
80
81 getchar ();
82 return 0;
83 }
84
85 //Multiplicar
86 Matriz* multiplicaMatrices (Matriz *a, Matriz *b){
87     Matriz *res;
88     unsigned int i, j, z;
89
90     res = rcCrearMatriz(a->nCol, b->nRen);
91
92     //Si el dato de nRen es diferente al dato de nCol no se podra realizar la matriz
93     //Ya que para realizar la multiplicacion de matrices necesitamos que el ultimo
94     //dato de nRen y nCol sean iguales para realizar el producto
95     if (a->nRen != b->nCol){
96
97         printf ("\n\nNo se puede realizar la multiplicacion de matrices\n\n");
98         exit (0);
99     }
100     else{
101         for (i=0; i<res->nRen; i++)
102         {
103             for (j=0; j<res->nCol; j++)
104             {
105                 *(res->data + i*res->nCol + j) = 0;
106                 for (z=0; z<res->nCol; z++)
107                 {
108                     *(res->data + i*res->nCol + j) += (*(a->data + i*a->nRen + z)) * (*(b->data + z*b->nCol + j));
109                 }
110             }
111         }
112     }
113
114     return res;
115 }
116
117 //Suma Matrices
118 Matriz* sumaMatrices (Matriz *a, Matriz *b){
119     Matriz *resultado;
120     unsigned int i, j;
121
122     resultado = rcCrearMatriz(a->nRen, a->nCol);
123
124     for (i=0; i<resultado->nRen; i++)
125     {
126         for (j=0; j<resultado->nCol; j++)
127         {
128             //Se recorre toda la matriz para ir sumando renglon-renglon o columna-columna
129             *(resultado->data + i*resultado->nCol + j) = *(a->data + i*a->nCol + j) + *(b->data + i*b->nCol + j);
130         }
131     }
132
133     return resultado;
134 }
135
136 //Liberar Memoria
137 void rcEliminarMatriz (Matriz *p){
138     //Elimanos memoria de los datos contenidos en la Matriz
139     free (p->data);
140     free (p);
141 }
142

```

```

143 //Mostrar Matriz
144 void rcMostarMatriz(Matriz *p){
145
146     //Recorrer renglones y columnas
147     unsigned int i, j;
148
149     for (i=0; i<p->nRen; i++)
150     {
151         printf ("\n");
152         for (j=0; j<p->nCol; j++)
153         {
154             printf("%g ", *(p->data + i*p->nCol + j));
155             //Dirección correspondiente del almacenado en i y j
156         }
157     }
158 }
159
160 //Capturar Matriz
161 void rcCapturaMatriz(Matriz *p){
162
163     //Recorrer renglones y columnas
164     unsigned int i, j;
165
166     for (i=0; i<p->nRen; i++)
167     {
168         for (j=0; j<p->nCol; j++)
169         {
170             printf("Dato [%u][%u]: ", i, j);
171             scanf("%f", p->data + i*p->nCol + j); fflush (stdin);
172             //Dirección correspondiente del almacenado en i y j
173         }
174     }
175 }
176
177 //Crear Matriz
178 Matriz* rcCrearMatriz (unsigned int nRen, unsigned int nCol){
179     Matriz *ptr; //Declarar nuestra variable en forma de apuntador
180
181     //Almacena datos de estructura Matriz de manera dinámica.
182     ptr = (Matriz *)malloc(sizeof(Matriz));
183
184     //Verificar si se almacenó la memoria
185     if (ptr == NULL){
186         printf ("Error al asignar memoria para la estructura [Matriz]");
187         exit (0);
188     }
189
190     ptr->nRen = nRen;
191     ptr->nCol = nCol;
192     ptr->data = (float *)malloc(nRen*nCol*sizeof(float));
193
194
195     return ptr;
196 }
197

```

Pruebas y Resultados:

Ahora que ya tenemos el problema planteado hay que implementar una solución de acuerdo con lo que se nos pide resolver en el reporte.

Iniciamos creando un nuevo archivo en nuestro editor de textos Notepad ++ el cual lo guardaremos con el nombre de T07.c ("En mi caso tiene el nombre de Clase13, ya que este ejercicio se planteó en la clase).

Comenzamos escribiendo nuestras librerías **"#include <stdio.h>"** y **"#include <stdlib.h>"** ya que esta última librería nos permitirá utilizar las operaciones de apuntadores junto con la memoria dinámica que se estará utilizando en nuestro programa.

Después de esto, declararemos una estructura con el nombre de **"Matriz"** la cual tendrá como argumento un apuntador de tipo **"float"** llamado **"data"** en cual será encargado de almacenar la dirección de un arreglo de datos bidimensionales, haciendo esto referencia a la matriz. Luego, declararemos dos variables más de tipo **"unsigned int"** que tendrán como nombre **"nRen"** – número de renglones, y **"nCol"** – número de columnas.

Comenzamos escribiendo nuestra función **"main"** donde contendrá nuestros bloques de código con ayuda de funciones que nos facilitarán en el desarrollo del programa.

Al comienzo, declaremos nuestra **"Matriz"** como parte de nuestra estructura de datos, en la cual se declarará cuatro variables a manera de apuntadores llamados **"a, b, c, d"** que nos ayudarán y se explicarán después. Ahora, declararemos dos variables de tipo **"unsigned int"** el cual tendrán como nombre **"ren"** – renglones y **"col"** – columnas, las cuales nos servirán para poder almacenar de que tamaño serán las matrices que el usuario quiere realizar.

Por este momento, ahora imprimiremos en pantalla "El numero de renglones y columnas para la matriz" y pediremos que el usuario ingrese estos valores en nuestras variables **"ren"** y **"col"** para poder crear nuestra matriz con esos valores y poder guardarlos en nuestro primer apuntador llamado **"*a"**.

NOTA: Función CrearMatriz

La función se llamará **"Matriz* rcCrearMatriz (unsigned int ren, unsigned int col)"** en la cual esta función estará dirigida a la estructura llamada **"Matriz"** para poder utilizar las variables que contiene; y también recibirá como parámetros el tamaño de la matriz.

Ahora declararemos un nuevo apuntador de tipo **"Matriz"**, llamada **"ptr"** el cual le asignaremos memoria dinámica con el tipo de dato que contiene **"Matriz"** de la siguiente manera:

- `ptr = (Matriz *)malloc(sizeof(Matriz));`

Por lo tanto, como nosotros asignamos memoria, hay que verificar si se asignó de manera correcta, ya que, si no ocurre esto tendremos problemas y no se podrá compilar.

Hacemos la verificación de memoria:

- ```
if (ptr == NULL){
 printf ("Error al asignar memoria para la estructura [Matriz]");
 exit (0);
}
```

Ahora viene lo más esencial de nuestro programa, como ya asignamos memoria a nuestra variable "**ptr**" tendremos que dirigirnos a las variables que contiene nuestra estructura de datos "**nRen**" y "**nCol**" y también nos dirigiremos a nuestro apuntador "**data**" que es la que almacenará las direcciones bidimensionales de la "**Matriz**" la cual también tenemos que asignarle memoria por las variables de "**nRen**" y "**nCol**" de un tipo "**float**", por último retornaremos el valor de "**ptr**".

NOTA: Declarar la función como prototipo: **Matriz\* rcCrearMatriz (unsigned int ren, unsigned int col);**

Como ya tenemos creada la dimensión de nuestra matriz con los parámetros necesarios, ahora tendremos que capturar los valores que contendrá nuestra primera "**Matriz [A]**", creando una función que nos ayudará a capturar dato por dato, esta función será vacía y como parámetros tendrá una matriz como apuntador llamada "**\*p**", que será la misma matriz que hemos creado.

Así será declarada y no olvidar ponerla como prototipo: **void rcCapturarMatriz (Matriz \*p);**

Dentro de esta función declaremos dos variables "**i, j**", las cuales nos ayudarán en recorrer los renglones y columnas con ayuda de nuestro ciclo "**for**".

Crearemos un primer ciclo "**for**" en el cual nos ayudará a recorrer los renglones, y después dentro de este mismo ciclo se creará otro "**for**" el cual nos ayuda a recorrer las columnas de la matriz, dentro de este último "**for**", pediremos que el usuario ingrese los valores cualesquiera que contenga la matriz con ayuda de nuestro apuntador "**p**" en la cual estará apuntando a los valores que están dentro de la estructura que declaramos desde un principio y ayudará a almacenar las direcciones de los valores ingresados.

De la siguiente manera:

- ```

for (i=0; i<p->nRen; i++)
{
    for (j=0; j<p->nCol; j++)
    {
        printf("Dato [%u][%u]: ", i, j);
        scanf("%f", p->data + i*p->nCol + j); fflush (stdin);
        //Dirección correspondiente del almacenado en i y j
    }
}

```

Ahora, crearemos una nueva función la cual nos ayudará en mostrar los valores que contiene nuestra matriz con ese formato.

Esta función será nuevamente vacía y tendrá como parámetro la “**Matriz *p**” que es la matriz que hemos creado al comienzo del programa.

Así será declarada y no olvidar ponerla como prototipo: **void rcMostrarMatriz (Matriz *p);**

Aquí de nuevo utilizaremos el mismo contenido que contenía la función “**CapturarMatriz**”

- ```

//Recorrer renglones y columnas
unsigned int i, j;

for (i=0; i<p->nRen; i++)
{
 printf ("\n");
 for (j=0; j<p->nCol; j++)
 {
 printf("%g ", *(p->data + i*p->nCol + j));
 //Dirección correspondiente del almacenado en i y j
 }
}

```

Ahora, el único dato aquí es que ya no volvemos a pedir datos solo los estaremos mostrando en su formato matricial recorriendo los renglones y columnas que contiene el apuntador “**p**” que está dirigiéndose a la estructura.

Teniendo en cuenta estas funciones, pediremos que el usuario ingrese la dimensión que desea manejar las matrices, después pasaremos estos valores a los apuntadores “**\*a**”, que será nuestra “**Matriz [A]**” y “**\*b**”, que será nuestra “**Matriz [B]**”, como se ve en el siguiente ejemplo:



- `printf ("Numero de renglones: ");`  
`scanf ("%u", &ren); fflush (stdin);`  
  
`printf ("Numero de columnas: ");`  
`scanf ("%u", &col); fflush (stdin);`  
  
`printf ("\nMATRIZ [A]\n");`  
`a = rcCrearMatriz(ren,col);`  
`rcCapturaMatriz(a);`  
  
`printf("\nLa matriz [A] capturada es: \n");`  
`rcMostarMatriz(a); //Mostrar Matriz`  
  
`printf ("\n\nMATRIZ [B]\n");`  
`b = rcCrearMatriz(ren,col);`  
`rcCapturaMatriz(b);`  
  
`printf("\nLa matriz [B] capturada es: \n");`  
`rcMostarMatriz(b); //Mostrar Matriz`

Ahora que ya tenemos nuestras dos matrices creadas con los valores que haya ingresado el usuario vamos a hacer las operaciones básicas de las matrices que son la suma y multiplicación.

Vamos a realizar la operación **“Suma”**

Por primera instancia, crearemos nuestra función de tipo **“Matriz”** de forma de apuntador, con el identificador de **“rcsumaMatrices (Matriz \*a, Matriz \*b)”**, en el cual como parámetros recibiremos las dos matrices con los datos que ingresó el usuario.

Dentro como argumento de nuestra función, declararemos una variable de tipo **“Matriz”** a forma de apuntador llamada **“resultado”**, ahora esta variable le asignaremos el valor de la creación de la matriz, con los parámetros que contiene la matriz **“a”**, dirigidas a la estructura con el **“nRen y nCol”**.

Ahora, declararemos dos variables **“unsigned int i, j”** que nos ayudarán a recorrer las matrices por renglón y columna.

Crearemos nuestro primer ciclo, que nos ayudará a recorrer el número de renglones y el segundo nos ayuda a recorrer las columnas de nuestras matrices, ahora dentro de nuestro último ciclo, se concentra las direcciones que contiene nuestra estructura de datos tanto de **“nRen y nCol”**, esto será igual a la suma de las direcciones que contiene la matriz **“a”** y la matriz **“b”** por medio del movimiento que contendrá en la parte de las columnas con ayuda de recorrer los renglones, de la siguiente manera se verá como se maneja este algoritmo:

- ```
for (i=0; i<resultado->nRen; i++)
{
    for (j=0; j<resultado->nCol; j++)
    {
        //Se recorre toda la matriz para ir sumando columna-columna
        *(resultado->data + i*resultado->nCol + j) = *(a->data + i*a->nCol +
        j) + *(b->data + i*b->nCol + j);
    }
}
```

Por último, tendremos que retornar “**resultado**”, que contendrá el valor de la matriz ya sumada.

No olvidar de poner el prototipo de la función: **Matriz* rcsomaMatrices (Matriz *a, Matriz *b);**

Ahora, para imprimir esta suma de matrices la guardaremos en nuestro apuntador “***c**”, el cual le asignaremos como valor la “**rcsomaMatrices(a,b)**”, ahora para imprimir este valor de nuestra suma nos ayudaremos de nuestra función “**rcMostrarMatriz**”, el cual únicamente le pasaremos como parámetro el apuntador “**c**”, el cual ya nos mostrará la adición de la matriz.

Vamos a realizar la operación “**Multipliación**”

Por primera instancia, crearemos nuestra función de tipo “**Matriz**” de forma de apuntador, con el identificador de “**rcmultiplicaMatrices (Matriz *a, Matriz *b)**”, en el cual como parámetros recibiremos las dos matrices con los datos que ingresó el usuario.

Dentro como argumento de nuestra función, declararemos una variable de tipo “**Matriz**” a forma de apuntador llamada “**res**”, ahora esta variable le asignaremos el valor de la creación de la matriz, con los parámetros que contiene la matriz “**a**”, dirigidas a la estructura con el “**nRen**” y la matriz “**b**” dirigida a la estructura “**nCol**”.

Ahora, declararemos dos variables “**unsigned int i, j, z**” que nos ayudarán a recorrer las matrices por renglón y columna.

Pero antes de realizar la operación producto de nuestras matrices hay que realizar primero una verificación, ya que en álgebra lineal se entiende que para multiplicar dos matrices hay que verificar que la columna de la primera matriz tiene que ser igual al renglón de la segunda matriz, si esto se cumple, se realiza la operación producto, de lo contrario no se podrá realizar esta multiplicación.

Si se cumple la condición de Renglón-Columna, vamos a realizar nuestros ciclos para recorrer la matriz por la parte de los renglones y columnas, y comenzaremos nuestra dirección de la matriz por medio del apuntador “**res**” en ceros para empezar a realizar la multiplicación por cada renglón-columna.

Por último, se usará el ciclo para realizar la operación producto correspondiente avanzando por esta vez en el número de Columnas.

- ```

//Si el dato de nRen es diferente al dato de nCol no se podra realizar la matriz
//Ya que para realizar la multiplicacion de matrices necesitamos que el ultimo
//dato de nRen y nCol sean iguales para realizar el producto

if (a->nRen != b->nCol){

 printf ("\n\nNo se puede realizar la multiplacacion de matrices\n\n");
 exit (0);

}else{
 for (i=0; i<res->nRen; i++)
 {
 for (j=0; j<res->nCol; j++)
 {
 *(res->data + i*res->nCol + j) = 0;
 for (z=0; z<res->nCol; z++)
 {
 *(res->data + i*res->nCol + j) += (*(a->data + i*a->nRen + z)) * (*(b->data
 + z*b->nCol + j));
 }
 }
 }
}

```

Por último, tendremos que retornar “**res**”, que contendrá el valor de la matriz ya multiplicada.

No olvidar de poner el prototipo de la función: **Matriz\* rcmultiplicaMatrices (Matriz \*a, Matriz \*b);**

Ahor, para imprimir este producto de matrices la guardaremos en nuestro apuntador “**\*d**”, el cual le asignaremos como valor la “**rcmultiplicaMatrices(a,b)**”, ahora para imprimir este valor de nuestro producto nos ayudaremos de nuestra función “**MostrarMatriz**”, el cual únicamente le pasaremos como parámetro el apuntador “**d**”, el cual ya nos mostrará la multiplicación de la matriz.

Para finalizar nuestro código, tenemos que eliminar los datos contenidos de la “**Matriz**”, creando una función vacía en la cual tendrá como nombre “**rcEliminarMatriz(Matriz \*p)**”, como parámetro le pasaremos la matriz que se creó, dentro de esta función liberaremos “**data**” el cual está dirigido a nuestra estructura que almacena la dirección de los datos bidimensionales y por último la creación de la matriz.

Teniendo esta función trabajando, debemos eliminar memoria de los apuntadores “**\*a, \*b, \*c, \*d**”, ya que no volveremos a usar sus datos.

Con esto finalizamos nuestro bloque de código, y ahora nos toca probarlo para observar su funcionamiento.

Para comprobar que nuestro programa realiza lo pedido, abriremos nuestro “**CMD**” e ingresaremos hasta la carpeta donde se tiene guardado el archivo; en mi caso se encuentra en la dirección:

- *C:\Users\rmanr\Documents\3\_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS*

Y con el comando <dir> veremos que el archivo se ha guardado de manera satisfactoria.

```

Símbolo del sistema
Microsoft Windows [Versión 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

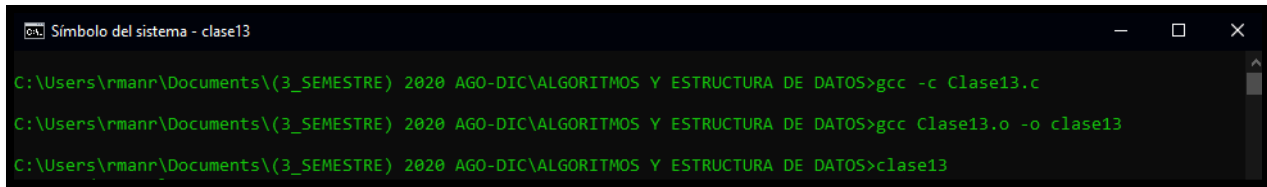
C:\Users\rmanr>cd C:\Users\rmanr\Documents\3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS

C:\Users\rmanr\Documents\3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>dir
El volumen de la unidad C es OS
El número de serie del volumen es: C6F9-7B98

Directorio de C:\Users\rmanr\Documents\3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS
18/09/2020 09:26 p. m. <DIR> .
18/09/2020 09:26 p. m. <DIR> ..
21/09/2020 05:38 p. m. 1,607 Clase10.c
10/09/2020 05:47 p. m. 46,330 clase10.exe
10/09/2020 05:47 p. m. 1,622 Clase10.o
15/09/2020 06:32 p. m. 3,451 Clase11.c
15/09/2020 06:40 p. m. 47,050 clase11.exe
15/09/2020 06:40 p. m. 2,843 Clase11.o
18/09/2020 02:18 p. m. 5,270 Clase12.c
18/09/2020 02:18 p. m. 48,242 clase12.exe
18/09/2020 02:18 p. m. 3,599 Clase12.o
18/09/2020 09:25 p. m. 4,231 Clase13.c
18/09/2020 09:26 p. m. 48,158 clase13.exe
18/09/2020 09:26 p. m. 3,526 Clase13.o

```

Ahora tendremos que compilar nuestro código con los siguientes comandos que se muestran en la imagen:

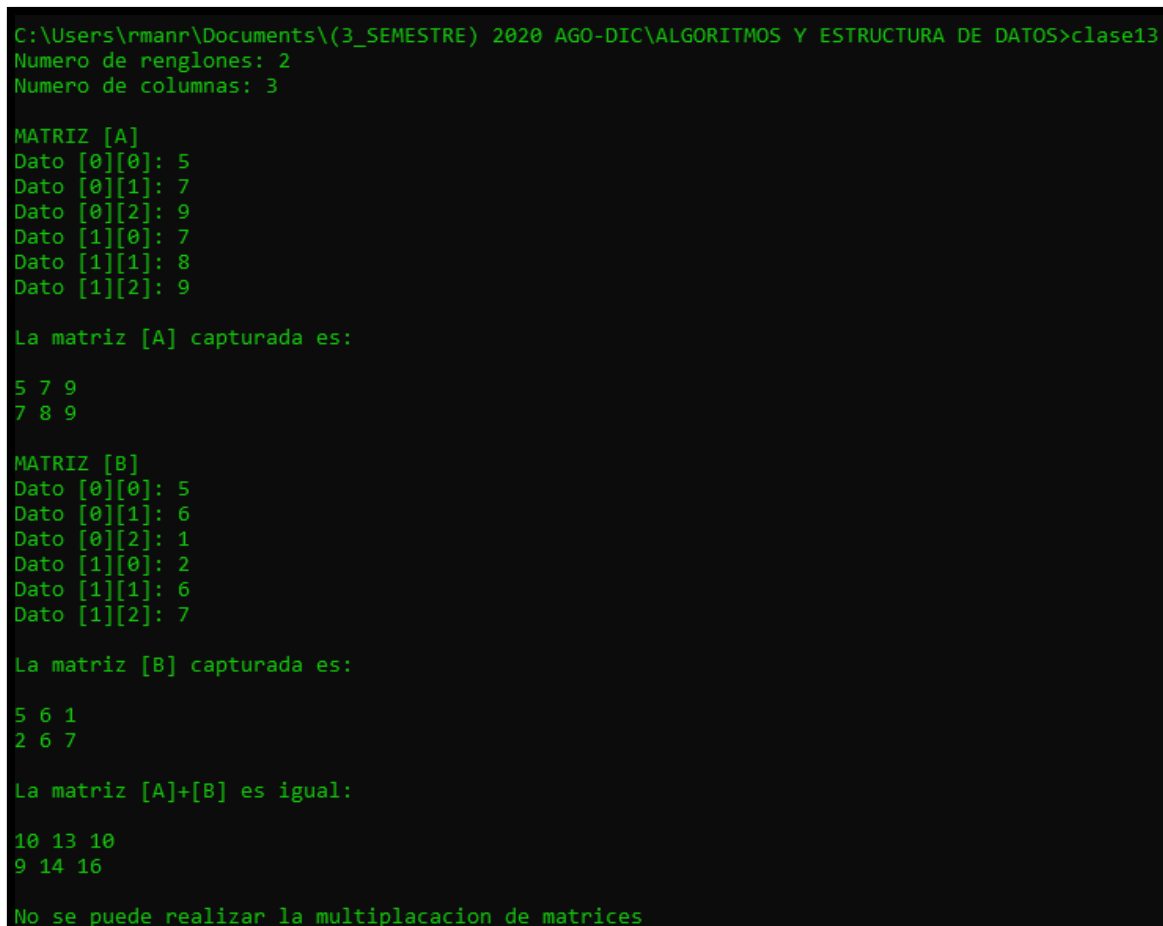


```
Símbolo del sistema - clase13
C:\Users\rmanr\Documents\3_SEMESTRE 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>gcc -c Clase13.c
C:\Users\rmanr\Documents\3_SEMESTRE 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>gcc Clase13.o -o clase13
C:\Users\rmanr\Documents\3_SEMESTRE 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>clase13
```

Si todo salió bien, solamente dará líneas de salto significando que todo el proceso de compilación y enlazamiento salió bien; de lo contrario, escribiste algún comando mal o tu código tiene algún error de sintaxis.

Ahora ejecutaremos nuestro programa para realizar las pruebas y observar si es lo que se quiso resolver desde un principio:

Observaremos el caso en que **no** se puede realizar una multiplicación de matrices:



```
C:\Users\rmanr\Documents\3_SEMESTRE 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>clase13
Numero de renglones: 2
Numero de columnas: 3

MATRIZ [A]
Dato [0][0]: 5
Dato [0][1]: 7
Dato [0][2]: 9
Dato [1][0]: 7
Dato [1][1]: 8
Dato [1][2]: 9

La matriz [A] capturada es:

5 7 9
7 8 9

MATRIZ [B]
Dato [0][0]: 5
Dato [0][1]: 6
Dato [0][2]: 1
Dato [1][0]: 2
Dato [1][1]: 6
Dato [1][2]: 7

La matriz [B] capturada es:

5 6 1
2 6 7

La matriz [A]+[B] es igual:

10 13 10
9 14 16

No se puede realizar la multiplacacion de matrices
```

Primero el programa nos preguntara de que tamaño queremos que crear nuestra matriz, nosotros le dimos de valor de 2 en la parte de los renglones y el valor de 3 en las columnas.

Ahora, para la matriz “**A**”, debemos ingresar los valores que nosotros deseamos en cada posición de la matriz, por ejemplo:

```
Dato [0][0]: 5
Dato [0][1]: 7
Dato [0][2]: 9
Dato [1][0]: 7
Dato [1][1]: 8
Dato [1][2]: 9
```

Después de ingresar los valores, nos mostrará cómo se ve en forma matricial los datos.

Ahora, para la matriz “**B**”, debemos ingresar los valores que nosotros deseamos en cada posición de la matriz, por ejemplo:

```
Dato [0][0]: 5
Dato [0][1]: 6
Dato [0][2]: 1
Dato [1][0]: 2
Dato [1][1]: 6
Dato [1][2]: 7
```

Después de ingresar los valores, nos mostrará cómo se ve en forma matricial los datos.

Ahora se mostrará la operación suma de matrices:

Matriz A+B:

$$\begin{pmatrix} 5 & 7 & 9 \\ 7 & 8 & 9 \end{pmatrix} + \begin{pmatrix} 5 & 6 & 1 \\ 2 & 6 & 7 \end{pmatrix} = \begin{pmatrix} 5+5 & 7+6 & 9+1 \\ 7+2 & 8+6 & 9+7 \end{pmatrix} = \begin{pmatrix} 10 & 13 & 10 \\ 9 & 14 & 16 \end{pmatrix}$$

Para el producto de la Matriz A\*B observamos que nos lanzó un mensaje en pantalla en que la “no se puede realizar la multiplicación de matrices” ya que el valor del renglón es diferente al valor de la columna y así no se podrá realizar esta operación.

Observaremos el caso en que **sí** se puede realizar una multiplicación de matrices:

```
C:\Users\rmanr\Documents\3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>clase13
Numero de renglones: 2
Numero de columnas: 2

MATRIZ [A]
Dato [0][0]: 5
Dato [0][1]: 9
Dato [1][0]: 7
Dato [1][1]: 8

La matriz [A] capturada es:

5 9
7 8

MATRIZ [B]
Dato [0][0]: 6
Dato [0][1]: 7
Dato [1][0]: 5
Dato [1][1]: 3

La matriz [B] capturada es:

6 7
5 3

La matriz [A]+[B] es igual:

11 16
12 11

La matriz [A]*[B] es igual:

75 62
82 73

Presione <ENTER> para salir
```

Primero el programa nos preguntara de que tamaño queremos que crear nuestra matriz, nosotros le dimos de valor de 2 en la parte de los renglones y el valor de 2 en las columnas.

Ahora, para la matriz “**A**”, debemos ingresar los valores que nosotros deseamos en cada posición de la matriz, por ejemplo:

```
Dato [0][0]: 5
Dato [0][1]: 9
Dato [1][0]: 7
Dato [1][1]: 8
```

Después de ingresar los valores, nos mostrará cómo se ve en forma matricial los datos.

Ahora, para la matriz “**B**”, debemos ingresar los valores que nosotros deseamos en cada posición de la matriz, por ejemplo:

Dato [0][0]: 6  
 Dato [0][1]: 7  
 Dato [1][0]: 5  
 Dato [1][1]: 3

Después de ingresar los valores, nos mostrará cómo se ve en forma matricial los datos.

Ahora se mostrará la operación suma de matrices:

Matriz A+B:

$$\begin{pmatrix} 5 & 9 \\ 7 & 8 \end{pmatrix} + \begin{pmatrix} 6 & 7 \\ 5 & 3 \end{pmatrix} = \begin{pmatrix} 5+6 & 9+7 \\ 7+5 & 8+3 \end{pmatrix} = \begin{pmatrix} 12 & 16 \\ 12 & 14 \end{pmatrix}$$

Para la multiplicación de la matriz A\*B: se mostrará como:

Matriz A\*B:

$$\begin{pmatrix} 5 & 9 \\ 7 & 8 \end{pmatrix} * \begin{pmatrix} 6 & 7 \\ 5 & 3 \end{pmatrix} = \begin{pmatrix} 5*6+9*5 & 5*7+9*3 \\ 7*6+8*5 & 7*7+8*3 \end{pmatrix} = \begin{pmatrix} 75 & 62 \\ 82 & 73 \end{pmatrix}$$

Con esto queda resuelto nuestro problema que se planteó desde un principio.

Como conclusión de este problema, sigue siendo muy interesante la forma en que nosotros podemos trabajar con apuntadores y memoria dinámica y ver su sencillez y elegancia que se utiliza en este código para resolver “suma y multiplicación” de matrices, utilizando una estructura de datos y funciones que nos apoyan a resolver con más sencillez y precisión las líneas de código que escribimos.

Espero ver más ejemplos que se le pueden dar haciendo uso de estos temas, además de poder seguir yo empleando más ejemplos que pueda realizar para mis programas.