	Carátula para entrega de prácticas	
Facultad de Ingeniería		Laboratorio de docencia

Laboratorio de computación Salas A y B

Profesor: Dr. Ismael Everardo Bárcenas Patiño

Asignatura: Estructuras de datos y algoritmos I

Grupo: 3

No. de práctica: 1

Integrantes:
10 Gurrión Aquino Carlos Ángel
14 León Ruiz Eduardo
16 Macías Niño Carmen Violeta
17 Marroquín García Ricardo
21 Montaño Torres Rodolfo Santiago

No. de equipo de cómputo empleado:

No. de brigada: 7

Semestre: 2022-1

Fecha de entrega: 7 de septiembre de 2021

Observaciones:

Calificación:

Rúbrica de evaluación:

100	El programa cumple con todos los requerimientos
80-99	El programa cumple con la mayoría de los requerimientos
60-79	El programa cumple con algunos de los requerimientos
50-59	La lógica del programa es correcta, pero no corre o se cuelga

Práctica 1 | Aplicaciones de arreglos

Problema a resolver:

I. Escribir un programa en C que cumpla con las siguientes características:

- lea una matriz de enteros $A \in \mathcal{M}_{n\times m}(\mathbb{Z})$, donde $n, m \in \mathbb{N}$;
- imprima la matriz

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,m} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,0} & a_{n,1} & \cdots & a_{n,m} \end{pmatrix};$$

- lea otra matriz de enteros $B \in \mathcal{M}_{m\times k}(\mathbb{Z})$, donde $k \in \mathbb{N}$;
- imprima la matriz

$$B = \begin{pmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,k} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,0} & b_{m,1} & \cdots & b_{m,k} \end{pmatrix};$$

- imprima el producto $A \times B \in \mathcal{M}_{n\times k}(\mathbb{Z})$;
- imprima el producto

$$A \times B = \begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,k} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,0} & c_{n,1} & \cdots & c_{n,k} \end{pmatrix},$$

donde

$$\begin{aligned} c_{0,0} &= a_{0,0} * b_{0,0} + a_{0,1} * b_{1,0} + \cdots + a_{0,m} * b_{m,0} \\ c_{0,1} &= a_{0,0} * b_{0,1} + a_{0,1} * b_{1,1} + \cdots + a_{0,m} * b_{m,1} \\ &\vdots \\ c_{0,k} &= a_{0,0} * b_{0,k} + a_{0,1} * b_{1,k} + \cdots + a_{0,m} * b_{m,k} \\ c_{1,0} &= a_{1,0} * b_{0,0} + a_{1,1} * b_{1,0} + \cdots + a_{1,m} * b_{m,0} \\ &\vdots \\ c_{n,k} &= a_{n,0} * b_{0,k} + a_{n,1} * b_{1,k} + \cdots + a_{n,m} * b_{m,k}. \end{aligned}$$

Código solución:

Enlace al código en compilador en línea: https://onlinegdb.com/kNcq9hrkB

```c
#include <stdio.h>

//Declaración de variables globales
  //Dos espacios en memoria que actúan como iteradores en varias funciones
int i;
int j;

  //Cuatro espacios en memoria para almacenar el tamaño de las dos matrices
int m = 0;
int nA = 0;
int nB = 0;
int k = 0;

  //Apuntador usado en el tamaño del argumento: int (*M)[*h] de las funciones readArray y printArray
int* h = &nA;

  //Dos conjuntos de tres espacios de memoria contiguos para almacenar los nombres predefinidos para las matrices y sus dimensiones
char a[] = "mnA";
char b[] = "nkB";


//Declaración de funciones
void progDescription(); //Imprime la descripción del programa
void sizeArray(char* x, int* u, int* v);  //Imprime una solicitud al usuario para que ingrese el tamaño de una matriz
void readSize(char* y, int* p, int z);  //Lee un entero positivo que corresponde al tamaño de una dimensión de una matriz
void readArray(int (*M)[*h], char* x, int* u, int* v);  //Lee los elementos de una matriz
void printArray(int (*M)[*h], char* x, int* u, int* v);  //Imprime los elementos de una matriz
void matrixMultiplication(int (*X)[nA], int (*Y)[k]);  //Multiplica dos matrices e imprime la matriz producto


//Función principal
float main()
{
  progDescription();

  sizeArray(a, &m, &nA);  //Se piden m y n
  int A[m][nA];  //Se crea un arreglo bidimensional de tamaño m x n

  readArray(A, a, &m, &nA);  //Se piden los elementos de la matriz


  sizeArray(b, &nB, &k);  //Se piden n y k
  int B[nB][k];  //Se crea un arreglo bidimensional de tamaño n x k
  h = &k;  //Se modifica la dirección en memoria a la que apunta el apuntador h para que readArray se ejecute con el valor de k  : int (*M)[k]
  readArray(B, b, &nB, &k);  //Se piden los elementos de la matriz


  h = &nA;  //Se vuelve a modificar el apuntador h para que apunte de nuevo a la dirección de nA  : int (*M)[nA]
  printArray(A, a, &m, &nA);

  h = &k;  //Se modifica el apuntador h para que apunte a k  : int (*M)[k]
  printArray(B, b, &nB, &k);

  matrixMultiplication(A, B);  //Multiplicamos y mostramos el resultado en una misma función

  return 3.141592;
}
```

```c
//Funciones secundarias
void progDescription()
{
  puts("PROGRAMA PARA CALCULAR EL PRODUCTO DE DOS MATRICES\n");
  printf("Este programa recibe como entrada dos matrices ");
  printf("%c y %c de tamaño %c x %c y %c x %c respectivamente ", *(a + 2), *(b + 2), *a, *(a + 1), *b, *(b + 1));
//Nótese que *(a + i) = a[i]
  printf("(donde %c, %c, %c son enteros positivos) cuyos elementos son números enteros. ", *a, *(a + 1), *(b +
1));
  printf("El programa multiplica ambas matrices %c y %c.  Y como datos de salida, se imprimen ", *(a + 2), *(b +
2));
  printf("las matrices factores y la matriz producto.");
}


void sizeArray(char* x, int* u, int* v)
{
  printf("\n\nIngrese el tamaño %c x %c de la matriz %c:\n", *x, *(x + 1), *(x + 2));

  readSize(x, u, 0);

  readSize(x, v, 1);
}


void readSize(char* y, int* p, int z)
{
  printf("\t%c = ", *(y + z));  //z toma valores de 0 y 1; así que *(y + 0) => m   *(y + 1) => n   cuando y es
igual al arreglo a
  scanf("%d", p);

  //Validación de datos
  while(nB != 0 && nB != nA)  //Valida que el número de columnas de A sea el número de filas de B
  {
    printf("\tWarning: El número de columnas de %c debe ser igual que el número de filas de %c. Ingrese un valor
válido.\n", *(a + 2), *(b + 2));
    printf("\tn = ");
    scanf("%d", &nB);
  }
  while(*p < 1)  //Valida que el entero ingresado sea positivo
  {
    printf("\tWarning: El valor de %c debe ser un entero positivo.\n", *(y + z));
    printf("\t%c = ", *(y + z));
    scanf("%d", p);
  }
}


void readArray(int (*M)[*h], char* x, int* u, int* v)  //Nótese que se recibe como argumento int (*M)[*h], que es
un apuntador a un arreglo de dimensión *h
{
  printf("\nIngrese los %d elementos de la matriz %c fila por fila.\n", (*u) * (*v), *(x + 2));

  for (i = 0; i < *u; i ++)  //for que recorre las filas de la matriz
  {
    printf("\tIngrese los %d enteros de la fila %d:\n", *v, i + 1);
    for (j = 0; j < *v; j ++)  //for que recorre las columnas de la matriz
    {
      printf("\t%c%d%d: ", *(x + 2), i + 1, j + 1);
      scanf("\t\t%d", *(M + i) + j);  //Se almacena el valor en &M[i][j]
    }
  }
}


void printArray(int (*M)[*h], char* x, int* u, int* v)
{
  printf("\n\tMatriz %c:\n", *(x + 2));
```

```
  for (i = 0; i < *u; i ++)  //for que recorre las filas de la matriz
  {
    printf("\t\t\t");
    for (j = 0; j < *v; j ++)  //for que recorre las columnas de la matriz
    {
      printf("%5d  ", *(*(M + i) + j));  //Se imprime M[i][j]. Nótese que *(*(M + i) + j) = M[i][j]
    }
    printf("\n");
  }
}


void matrixMultiplication(int (*X)[nA], int (*Y)[k])
{
  //Variables locales
  int r;  //Iterador
  int result;  //Variable que almacena cada elemento de la matriz producto

  puts("\n\tMatriz A x B:\n");

  for(i = 0; i < m; i ++)  //for que recorre las filas de la matriz producto (filas de A)
  {
    printf("\t\t\t");
    for(j = 0; j < k; j ++)  //for que recorre las columnas de la matriz producto (columnas de B)
    {
      for(result = 0, r = 0; r < nA; r ++)  //for que recorre el número de columnas de A (o filas de B) | result
se inicializa en 0 en cada iteración de j
      {
        result += (*(*(X + i) + r)) * (*(*(Y + r) + j));  //Se almacena el resultado en result
      }
      printf("%5d  ", result);  //Se imprime result
    }
    printf("\n");
  }
}
```