Problemas de Álgebra con C

1. El siguiente programa debe calcular la suma de dos matrices definidas en la función `main()`. Definir la función `suma()` para que funcione correctamente.

```
#include <stdio.h>

int main(int argc, char** argv) {
    double A[3][3]={{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}};
    double B[3][3]={{1.,0.,0.},{0.,1.,0.},{0.,0.,1.}};
    double C[3][3];
    suma(A,*B,C[0]);
    printf("La suma de las matrices A y B es\n%g\t%g\t%g\n%g\t%g\t%g\n%g\t%g\t%g\n",
    C[0][0],C[0][1],C[0][2],C[1][0],C[1][1],C[1][2],C[2][0],C[2][1],C[2][2]);
    return 0;
}
```

Solución:

```
void suma(double A[][3], double *B, double *C) {
      int i, j;
      for (i=0; i<3; i++) {
         for (j=0; j<3; j++)
             *(C+3*i+j)=A[i][j]+*(B+3*i+j);
      }
      return;
}
```

2. En un determinado programa estamos trabajando con vectores en el espacio tridimensional definidos a través de estructuras del siguiente modo:

```
struct Vector {
    double x, y, z;
};
typedef struct Vector vector;
```

Defínase una función que calcule (y devuelva) el producto vectorial de dos vectores cualesquiera, introducidos como argumentos de la función.

Solución:

```
Vector producto_vectorial(Vector a, Vector b) {
    Vector c;
    c.x=  a.y*b.z - a.z*b.y;
    c.y= -a.x*b.z + a.z*b.x;
    c.z=  a.x*b.y - a.y*b.x;
    return c;
}
```

3. Supongamos que estamos trabajando con vectores en el espacio tridimensional definidos a través de estructuras del siguiente modo:

```
struct Vector {
    double x, y, z;
};
typedef struct Vector vector;
```

Definir una función, que llamaremos `cambioDeBase()`, que transforme (sin devolver nada) las componentes del vector **r** al aplicar el cambio de base dado por la matriz **C** de números reales. Las nuevas componentes estarán dadas por el producto $\mathbf{C} \cdot \mathbf{r}$. El vector **r** está declarado como:

```
vector r;
```

y la función será llamada desde el programa principal del siguiente modo:

```
cambioDeBase(&r, C);
```

Solución:

```
void cambioDeBase(vector *r, double C[][3]) {
    int j;
    vector aux;
    aux.x=C[0][0]*(r->x)+C[0][1]*(r->y)+C[0][2]*(r->z);
    aux.y=C[1][0]*(r->x)+C[1][1]*(r->y)+C[1][2]*(r->z);
    aux.z=C[2][0]*(r->x)+C[2][1]*(r->y)+C[2][2]*(r->z);
    *r=aux;
}
```

4. Señale cuáles son las 5 líneas de código erróneas en la función siguiente, pensada para calcular el módulo de un vector, y en qué consisten dichos errores:

```
1 double f(x[4]) {
2     int i;
3     double y;
4     for(i=1; i<=4; i++)
5         y+=y+x[i]*x[i];
6     return sqr(y);
7 }
```

Solución: Línea 1: falta declarar el tipo del vector `x[]`. Línea 3 (por ejemplo): falta inicializar el valor de `y`. Línea 4: el recorrido de `i` debe comenzar en `0`, y debe llegar hasta `3` (`i<4`). Línea 5: o bien se incrementa `y` (con `y+=`) o bien se calcula la nueva y a partir de la previa (`y=y+`), pero no ambas. Línea 6: la raíz cuadrada se calcula en C con la función `double sqrt(double)` de la `<math.h>`.

5. Complete los puntos suspensivos en las líneas indicadas en la siguiente función para que el resultado del cálculo sea el número de celdas activas (con valor #definido ON) en las dos matrices pasadas, ambas de tamaño NxN (valor también #definido).

```
int coincidenciasON(int A[N][N], int B[N][N]) {
    int i,j, c=0;
    for(i=...) {
        for(j=...) {
            if( A[i][j] ... )
```

```
                c++;
            }
        }
        return c;
    }
```

Solución:

```
    int coincidenciasON(int A[N][N], int B[N][N]) {
        int i,j, c=0;
        for(i=0; i<N; i++) {
            for(j=0; j<N; j++) {
                if( A[i][j]==ON && B[i][j]==ON )
                    c++;
            }
        }
        return c;
    }
```

6. Indique qué guarda la siguiente función en el archivo.

```
void comoCSV(char* fn, char** cols, int n, int m, double** vals) {
    FILE *f=fopen(fn, "wt");
    int i, j;
    fprintf(f, "%s", cols[0]);
    for(j=0; j<m; j++) {
        fprintf(f, ",%s", cols[j]);
    }
    fprintf(f, "\n");
    for(i=0; i<n; i++) {
        fprintf(f, "%g", vals[i][0]);
        for(j=1; j<m; j++) {
            fprintf(f, ", %g", vals[i][j]);
        }
        fprintf(f, "\n");
    }
    fclose(f);
}
```

Solución: Guarda, en la primera línea, las cadenas que contiene `cols`, separadas por comas (los nombres de las columnas). En las restantes líneas del archivo, guarda los valores de la matriz `vals` fila a fila, separados por comas.

7. Escriba una función que, dada una matriz **A** y un vector(fila) **v**, calcule un vector $\mathbf{w} = \mathbf{v} \cdot \mathbf{A}$. Además de **v** y **A**, qué otros parámetros requeriría la función, en el caso general? [**Nota:** considerar que la matriz **A** nunca tendrá más de `NMAX` filas o columnas; considerar también que es responsabilidad del programador que use la función que los argumentos de ésta sean los adecuados para realizar el cálculo]

Solución: Para calcular el producto, es necesario conocer el número de filas y columnas de **A**. Además, el número de filas de **A** debe coincidir con la dimensión del vector **v**. El vector retornado, tendrá como dimensión el número de columnas de **A**.

```
void pvecMat(double v[], int nf, int nc, double A[][NMAX], double w[]) {
    double s;
    double i,j;
    for(j=0; j<nc; j++) {
        s=0.0;
        for(i=0; i<nf; i++) {
            s=s+v[i]*A[i][j];
        }
    w[j]=s;
    }
}
```

8. La rotación de cualquier vector del espacio $\mathbb{R}^3$ Euclídeo descrito mediante la base canónica cartesiana $\{\mathbf{i},\mathbf{j},\mathbf{k}\}$ un ángulo α alrededor de cada uno de los tres ejes cartesianos X, Y o Z, viene dada respectivamente por las siguientes matrices de cambio de base:

$$R_X(\alpha)=\begin{pmatrix}1 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha\\ 0 & \sin\alpha & \cos\alpha\end{pmatrix},\quad R_Y(\alpha)=\begin{pmatrix}\cos\alpha & 0 & \sin\alpha\\ 0 & 1 & 0\\ -\sin\alpha & 0 & \cos\alpha\end{pmatrix},\quad R_Z(\alpha)=\begin{pmatrix}\cos\alpha & -\sin\alpha & 0\\ \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{pmatrix}$$

Por ejemplo, las nuevas componentes $\mathbf{r}'=(x',y',z')$ resultantes de la rotación del vector $\mathbf{r}=(x,y,z)$ un ángulo α alrededor el eje Z, serán el resultado de $\mathbf{r}'=R_Z(\alpha)\mathbf{r}$, que en componentes tiene la forma:

$$\begin{aligned}x' &= x\cos\alpha - y\sin\alpha\\ y' &= x\sin\alpha + y\cos\alpha\\ z' &= z\end{aligned}$$

El objetivo de este ejercicio es diseñar un programa, que llamaremos `rotacion.c`, que realice la rotación de un vector cualquiera alrededor de alguno de los ejes coordenados. Las componentes del vector, el ángulo de giro y el eje respecto del cual se realizará el giro, que llamaremos 1 para el eje X, 2 para el eje Y y 3 para el eje Z, deberán ser introducidas por línea de comandos (como argumentos de la función `main`) cuando se ejecuta el programa. Por ejemplo, la rotación del vector $\mathbf{r}'=(0.2,1.4,-3.5)$ un ángulo de 30 alrededor del tercer eje Z se ejecutará como:

```
rotacion.exe 0.2 1.4 -3.5 30 3
```

La función `main` deberá leer estos datos y realizará el giro correspondiente. Cada uno de los tres giros posibles deberá ser definido en el programa mediante una función. La función `main` deberá escoger entre estas funciones y mandarle las componentes del vector y el ángulo de giro. La función correspondiente realizará el giro y devolverá las nuevas componentes del vector, que finalmente serán impresas en pantalla desde `main`. No es necesario en el examen escribir la definición completa de las tres funciones que realizan los giros, bastará con escribir de forma completa una de ellas. Las otras dos serán completamente equivalentes.

Solución: Ver código `rotacion.c`

9. Una matriz ortogonal A es una matriz cuadrada cuya matriz inversa A^{-1} coincide con su matriz traspuesta A^T: $A^{-1}=A^T$ o también $A\cdot A^T=\mathbf{1}$ donde $\mathbf{1}$ es la matriz identidad. Supongamos que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula la matriz inversa de una matriz cuadrada. Esta función está declarada del siguiente modo

```
void invierte_matriz(int orden, double A[][orden], double A_inverse[][orden])
```

donde `orden` es el orden de la matriz cuadrada A (es decir, si el orden es 2 la matriz será 2×2, y `A_inverse` es la matriz inversa calculada.

Escribir una función externa a `main` declarada del siguiente modo

```
int comprueba_matriz_ortogonal (int orden, double A[][orden])
```

que reciba como argumentos el orden de la matriz y la propia matriz, y que devuelva el valor 1 si la matriz introducida es ortogonal y 0 si no lo es.

Solución:

```
int comprueba_matriz_ortogonal (int orden, double A[][orden]) {
    int i, j, condicion;
    double AIN[orden][orden];
    invierte_matriz(orden, A, AIN);
    condicion=1;
    for(i=0; i<orden; i++) {
        for (j=0; j<orden; j++) {
            if (AIN[i][j]!=A[j][i]) condicion=0;
        }
    }
    return condicion;
}
```

10. Una matriz cuadrada de números complejos A es hermítica si cumple con la propiedad de que es igual a su traspuesta conjugada: $A = \overline{A^T}$, o en térmimos de sus componentes: $A_{ij} = \overline{A_{ji}}$, donde la barra denota el complejo conjugado. Supongamos que hemos definido un número complejo como un nuevo tipo de variable denominada `complex`, del siguiente modo:

```
struct num_complejo{
    double re, im;
};
typedef struct num_complejo complex;
```

Escribir una función en C (externa a `main`) que reciba en forma de argumento una matriz cuadrada de números complejos, y devuelva un 1 si la matriz es hermítica y un 0 si no lo es.

Solución:

```
int comprueba_matriz_hermitica (int N, complex A[][N]) {
    int i, j, condicion;
    condicion=1;
    for(i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            if (A[i][j].re !=A[j][i].re || A[i][j].im !=-(A[j][i].im))
                condicion=0;
        }
    }
    return condicion;
}
```

11. Una matriz hermítica A es una matriz cuadrada de números complejos que tiene la propiedad de que es igual a su traspuesta conjugada: $A = \overline{A^T}$, o en térmimos de sus componentes: $A_{ij} = \overline{A_{ji}}$, donde la barra denota el complejo conjugado.

Escribir una función en C (externa a `main`) que compruebe si una determinada matriz cuadrada de números complejos es hermítica. La función debe recibir en forma de argumentos dos matrices cuadradas de números reales, la primera con la parte real de A y la segunda con la parte imaginaria de A. La función debe devolver un 1 si la matriz es hermítica y un 0 si no lo es.

Solución:

```
int comprueba_matriz_hermitica (int N, double ReA[][N], double ImA[][N]) {
    int i, j, condicion;
    condicion=1;
    for(i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            if (ReA[i][j] != ReA[j][i] || ImA[i][j] != -ImA[j][i])
                condicion=0;
        }
    }
    return condicion;
}
```

12. Una matriz unitaria A es una matriz cuadrada de números complejos que tiene la propiedad de que su matriz inversa coincide con su matriz traspuesta conjugada: $A^{-1} = \overline{A^T}$, o en térmimos de sus componentes: $(A^{-1})_{ij} = \overline{A_{ji}}$, donde la barra denota el complejo conjugado.

    Supongamos que hemos definido un número complejo como un nuevo tipo de variable denominada `complex`, del siguiente modo:

```
struct num_complejo{
    double re, im;
};
typedef struct num_complejo complex;
```

    Supongamos también que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula la matriz inversa de cualquier matriz A de números complejos, de tamaño $N \times N$. Esta función está declarada del siguiente modo:

```
void invierte_matriz(int N, complex A[][N], complex Ainversa[][N])
```

    donde `Ainversa` es la matriz inversa calculada.

    Escribir una función externa a `main` que reciba en forma de argumento una matriz cuadrada de números complejos, y que devuelva el valor 1 si la matriz introducida es unitaria y 0 si no lo es.

    Solución:

```
int comprueba_matriz_unitaria (int N, complex A[][N])
{
    int i, j, condicion;
    complex Ainv[N][N];
    invierte_matriz(N, A, Ainv);
    condicion=1;
    for(i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            if (Ainv[i][j].re != A[j][i].re || Ainv[i][j].im !=-A[j][i
                ].im) condicion=0;
        }
    }
    return condicion;
}
```

13. Una matriz unitaria A es una matriz cuadrada de números complejos que tiene la propiedad de que su matriz inversa coincide con su matriz traspuesta conjugada: $A^{-1} = \overline{A^T}$, o en térmimos de sus

componentes: $(A^{-1})_{ij} = \overline{A_{ji}}$, donde la barra denota el complejo conjugado. Esto quiere decir que el producto de la matriz por su traspuesta conjugada es la matriz identidad I, esto es $A \cdot \overline{A^T} = I$, donde los elementos de la matriz I son todos 0 menos los de la diagonal que son 1.

Supongamos que hemos definido un número complejo como un nuevo tipo de variable denominada `complex`, del siguiente modo:

```
struct num_complejo{
    double re, im;
};
typedef struct num_complejo complex;
```

Supongamos también que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula el producto de dos matrices complejas cuadradas A y B, de tamaño $N \times N$. Esta función está declarada del siguiente modo:

```
void multiplica_matrices(int N, complex A[][N], complex B[][N], complex AxB[][N])
```

donde `AxB` es la matriz resultante del producto.

Escribir una función externa a `main` que reciba en forma de argumento una matriz cuadrada de números complejos, y que devuelva el valor 1 si la matriz introducida es unitaria y 0 si no lo es.

Solución:

```
int comprueba_matriz_unitaria (int N, complex A[][N])
{
    int i, j, condicion;
    complex TCA[N][N];      // matriz traspuesta conjugada de A
    complex Producto[N][N];
    for(i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            TCA[i][j].re=A[j][i].re;
            TCA[i][j].im=-A[j][i].im;
        }
    }
    multiplica_matrices(N, A, TCA, Producto);
    condicion=1;
    for(i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            if (i==j)
                {if (Producto[i][j].re != 1 || Producto[i][j].im !=0)
                    condicion=0;
            else
                {if (Producto[i][j].re != 0 || Producto[i][j].im !=0)
                    condicion=0;
        }
    }
    return condicion;
}
```

14. Supongamos que tenemos dos matrices A y B de números reales con igual tamaño. Se define el producto escalar de las dos matrices (A, B) como

$$(A, B) = \mathrm{tr}\left(A^T \cdot B\right)$$

En esta expresión A^T es la matriz traspuesta de A, que en térmimos de sus componentes esto quiere decir: $(A^T)_{ij} = A_{ji}$, el punto indica el producto entre matrices y tr es la función traza de una matriz, es decir, la suma de los elementos de su diagonal. Por lo tanto, para obtener el producto escalar de dos matrices, que es un número real, se multiplica la traspuesta de la primera matriz por la segunda matriz, y después se calcula la traza de la matriz resultante.

Supongamos que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula el producto de dos matrices cualquiera A y B, de tamaños $n \times p$ y $p \times m$, respectivamente. Esta función está declarada del siguiente modo:

```
void multiplica_matrices(int n, int p, int m,
                         double A[][p], double B[][m], double AxB[][m])
```

donde AxB es la matriz resultante del producto y tiene tamaño $n \times m$.

Escribir una función externa a main que reciba en forma de argumentos dos matrices con el mismo tamaño y devuelva su producto escalar, haciendo uso de la función anterior.

Solución:

```
double producto_escalar_matrices (int n, int m, double A[][m], double
    B[][m]) }
     int i,j;
     double AT[m][n];
     double AxB[m][m];
     double traza=0;
     for(i=0; i<n; i++) {
          for(j=0; j<m; j++) {
               AT[j][i]=A[i][j];
          }
     }
     multiplica_matrices(m, n, m, AT, B, AxB);
     for(i=0; i<m; i++) traza+=AxB[i][i];
     return traza;
}
```

15. Supongamos que tenemos dos matrices A y B de números reales con igual tamaño. Se define el producto escalar de las dos matrices (A, B) como

$$(A, B) = \mathrm{tr}\left(A \cdot B^T\right)$$

En esta expresión B^T es la matriz traspuesta de B, que en térmimos de sus componentes esto quiere decir: $(B^T)_{ij} = B_{ji}$, el punto indica el producto entre matrices y tr es la función traza de una matriz, es decir, la suma de los elementos de su diagonal. Por lo tanto, para obtener el producto escalar de dos matrices, que es un número real, se multiplica la primera matriz por la traspuesta de la segunda, y después se calcula la traza de la matriz resultante.

Supongamos que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula el producto de dos matrices cualquiera A y B, de tamaños $n \times p$ y $p \times m$, respectivamente. Esta función está declarada del siguiente modo:

```
void multiplica_matrices(int n, int p, int m, double A[][p],
              double B[][m], double AxB[][m])
```

donde AxB es la matriz resultante del producto y tiene tamaño $n \times m$.

Escribir una función externa a main que reciba en forma de argumentos dos matrices con el mismo tamaño y devuelva su producto escalar, haciendo uso de la función anterior.

Solución:

```
double producto_escalar_matrices (int n, int m, double A[][m], double
    B[][m]) {
     int i,j;
     double BT[m][n];
     double AxB[n][n];
     double traza=0;
     for(i=0; i<n; i++) {
          for(j=0; j<m; j++) {
               BT[j][i]=B[i][j];
          }
     }
     multiplica_matrices(n, m, n, A, BT, AxB);
     for(i=0; i<n; i++) traza+=AxB[i][i];
     return traza;
}
```

16. La norma de una matriz A de números reales de cualquier tamaño puede ser definida del siguiente modo:

$$\|A\| = \sqrt{\mathrm{tr}\,(A^T \cdot A)}$$

En esta expresión A^T es la matriz traspuesta de A, que en térmimos de sus componentes esto quiere decir: $(A^T)_{ij} = A_{ji}$, el punto indica el producto entre matrices y tr es la función traza de una matriz, es decir, la suma de los elementos de su diagonal. Por lo tanto, para obtener la norma de una matriz, que es un número real, se multiplica su matriz traspuesta por ella misma, después se calcula la traza de la matriz resultante, y finalmente se calcula la raíz cuadrada de este valor.

Supongamos que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula el producto de dos matrices cualquiera A y B, de tamaños $n \times p$ y $p \times m$, respectivamente. Esta función está declarada del siguiente modo:

```
void multiplica_matrices(int n, int p, int m, double A[][p], double B[][m],
                                                  double AxB[][m])
```

donde AxB es la matriz resultante del producto y tiene tamaño $n \times m$.

Escribir una función externa a main que reciba en forma de argumento una matriz y calcule su norma, haciendo uso de la función anterior.

Solución:

```
double norma_matriz (int n, int m, double A[][m]) {
     int i,j;
     double AT[m][n];
     double ATxA[m][m];
     double traza=0;
     for(i=0; i<n; i++) {
          for(j=0; j<m; j++) {
               AT[j][i]=A[i][j];
          }
     }
     multiplica_matrices(m, n, m, AT, A, ATxA);
     for(i=0; i<m; i++) traza+=ATxA[i][i];
     return sqrt(traza);
}
```

17. La norma de una matriz A de números reales de cualquier tamaño puede ser definida del siguiente modo:

$$\|A\| = \sqrt{\mathrm{tr}\,(A \cdot A^T)}$$

En esta expresión A^T es la matriz traspuesta de A, que en térmimos de sus componentes esto quiere decir: $(A^T)_{ij} = A_{ji}$, el punto indica el producto entre matrices y tr es la función traza de una matriz, es decir, la suma de los elementos de su diagonal. Por lo tanto, para obtener la norma de una matriz, que es un número real, se multiplica la matriz por su traspuesta, después se calcula la traza de la matriz resultante, y finalmente se calcula la raíz cuadrada de este valor.

Supongamos que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula el producto de dos matrices cualquiera A y B, de tamaños $n \times p$ y $p \times m$, respectivamente. Esta función está declarada del siguiente modo:

```
void multiplica_matrices(int n, int p, int m, double A[][p],
                                               double B[][m], double AxB[][m])
```

donde AxB es la matriz resultante del producto y tiene tamaño $n \times m$.

Escribir una función externa a main que reciba en forma de argumento una matriz y calcule su norma, haciendo uso de la función anterior.

Solución:

```
double norma_matriz (int n, int m, double A[][m]) {
    int i,j;
    double AT[m][n];
    double ATxA[m][m];
    double traza=0;
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            AT[j][i]=A[i][j];
        }
    }
    multiplica_matrices(m, n, m, AT, A, ATxA);
    for(i=0; i<m; i++) traza+=ATxA[i][i];
    return sqrt(traza);
}
```

18. Supongamos que tenemos dos matrices A y B de números reales con igual tamaño. Se define la distancia entre las dos matrices $d(A, B)$ como la norma de la matriz diferencia entre las dos matrices:

$$d(A, B) = \|A - B\|$$

La norma de una matriz C de números reales de cualquier tamaño se calcula como:

$$\|C\| = \sqrt{\mathrm{tr}\,(C^T \cdot C)}$$

En esta expresión C^T es la matriz traspuesta de C, que en térmimos de sus componentes esto quiere decir: $(C^T)_{ij} = C_{ji}$, el punto indica el producto entre matrices y tr es la función traza de una matriz, es decir, la suma de los elementos de su diagonal. Por lo tanto, para obtener la distancia entre dos matrices (que es un número real) primero debemos restarlas, después multiplicar la traspuesta de la matriz diferencia por ella misma, y finalmente calcular la raíz cuadrada de la traza de la matriz resultante.

Supongamos que tenemos una biblioteca de funciones en la que tenemos definida una función que calcula el producto de dos matrices cualquiera A y B, de tamaños $n \times p$ y $p \times m$, respectivamente. Esta función está declarada del siguiente modo:

```
void multiplica_matrices(int n, int p, int m, double A[][p],
                                               double B[][m], double AxB[][m])
```

donde AxB es la matriz resultante del producto y tiene tamaño $n \times m$.

Escribir una función externa a main que reciba en forma de argumentos dos matrices con el mismo tamaño y devuelva la distancia que existe entre ellas, haciendo uso de la función anterior.

Solución:

```
double distancia_entre_matrices (int n, int m, double A[][m], double B[][m]) {
    int i,j;
    double C[n][m];
    double CT[m][n];
    double CTxC[m][m];
    double traza=0;
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            C[i][j]=A[i][j]-B[i][j];
            CT[j][i]=C[i][j];
        }
    }
    multiplica_matrices(m, n, m, CT, C, CTxC);
    for(i=0; i<m; i++) traza+=CTxC[i][i];
    return sqrt(traza);
}
```

19. Escriba un programa que pida al usuario las coordenadas de dos vectores tridimensionales, las guarde en forma de estructuras

```
struct Vec3f {
        float x, y, z;
}
```

Y llame a una función (que también deberá implementar)

```
double angVec3f(struct Vec3f a, struct Vec3f b);
```

que determine el ángulo en grados que forman y lo devuelva al usuario imprimiéndolo por pantalla.

Nota: recuerde que el ángulo α formado por dos vectores $\boldsymbol{a}$, $\boldsymbol{b}$ viene dado por $\alpha = \arccos\left(\frac{\boldsymbol{a}\cdot\boldsymbol{b}}{\sqrt{(\boldsymbol{a}\cdot\boldsymbol{a})}\sqrt{(\boldsymbol{b}\cdot\boldsymbol{b})}}\right)$ y el producto escalar de dos vectores $\boldsymbol{a}\cdot\boldsymbol{b} = a_x b_x + a_y b_y + a_z b_z$.

Solución:

```
#include <stdio.h>
#include <math.h>

struct Vec3f {
    float x, y, z;
}

int main(int argc, char** argv) {
    struct Vec3f a, b;
    float alpha;
    printf("a= ");
```

```
    scanf("%f%f%f", &a.x, &a.y, &a.z);
    printf("b= ");
    scanf("%f%f%f", &b.x, &b.y, &b.z);
    alpha=angVec3f(a,b);
    printf("alpha = %g\n", alpha);
    return 0;
}

float angVec3f(struct Vec3f a, struct Vec3f b) {
    float ab, ma, mb, x;
    ab=a.x*b.x+a.y*b.y+a.z*b.z;
    ma=sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
    mb=sqrt(b.x*b.x+b.y*b.y+b.z*b.z);
    x=acos( ab/(ma*mb) );
    return (180/M_PI)*x;
}
```

20. Una matriz se puede guardar en forma de estructura

```
struct Matriz {
int nfilas, ncolumnas;
double *c_ij;
};
```

donde `nfilas` y `ncolumnas` guardan las dimensiones de la matriz, y el vector `c_ij` almacena los nfilas*ncolumnas números reales de las celdas de la matriz, en las posiciones dadas por i+nfilas*j, donde i es la fila de la celda, y j la columna. Escriba una función que, recibiendo la matriz por referencia (esto es, un puntero a una `struct Matriz`), retorne el valor del elemento de la matriz que tiene mayor valor absoluto.

Solución:

```
double max_matriz(struct Matriz *M) {
  int n, N=M->nfilas*M->ncolumnas;
  double xm=M->c_ij[0];
  for(n=1; n<N; n++) {
    if( abs(M->c_ij[n]) > abs(xm) ) {
      xm = M->c_ij[n];
    }
  }
  return xm;
}
```

21. La norma de una matriz A de números reales, de tamaño $n \times m$ (donde n es el número de filas y m el de columnas), puede ser calculada del siguiente modo:

$$\|A\| = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} (A_{ij})^2 \right)^{1/2}$$

donde A_{ij} es el elemento de la matriz que ocupa la fila i y la columna j. Escribir una función externa a `main` que reciba como argumentos el tamaño de una matriz y su primer elemento (este último deberá ser pasado "por referencia"), y que calcule la norma de la matriz. Por tanto, la función debe estar declarada como:

```
double calcula_norma_matriz(int n, int m, double *A);
```

Supongamos que hemos declarado nuestra matriz A en `main` del siguiente modo:

```
double A[2][3];
```

¿Cómo deberemos llamar a la función `calcula_norma_matriz` para que calcule la norma de la matriz A?

Solución:

```
double calcula_norma_matriz(int n, int m, double *A) {
    double norma=0;
    int i,j,ij;
    ij=0;
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            norma+=A[ij]*A[ij];
            ij++;
        }
    }
    return sqrt(norma);
}
```

Si consideramos:

```
double A[2][3]={{1,-2,3},{4,-5,6}};
```

la llamada a la función deberá ser:

```
norma=calcula_norma_matriz(2, 3, A[0]);
```

22. La p-norma (con p real ≥ 1) de una matriz A de números reales, de tamaño $n \times m$ (donde n es el número de filas y m el de columnas), puede ser calculada del siguiente modo:

$$\|A\| = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} |A_{ij}|^p \right)^{1/p}$$

donde A_{ij} es el elemento de la matriz que ocupa la fila i y la columna j.

Escribir una función externa a `main`, de tipo `void`, que reciba como argumentos una matriz y su tamaño, la clase de norma p que queremos aplicar, y que calcule su p-norma. El resultado obtenido se debe retornar "por referencia" en una variable tipo `double` que también es recibida por la función como argumento.

Solución:

```
void p_norma_matriz(int n, int m, double A[][m], double p, double* Norma) {
    double norma=0;
    int i,j;
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            norma+=pow(fabs(A[i][j]),p);
        }
    }
```

```
    *Norma=pow(norma,1./p);
    return;
}
```

23. La norma de una matriz A de números reales, de tamaño $n \times m$ (donde n es el número de filas y m el de columnas), puede ser calculada del siguiente modo:

$$\|A\| = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} (A_{ij})^2 \right)^{1/2}$$

donde A_{ij} es el elemento de la matriz que ocupa la fila i y la columna j.

Escribir una función externa a `main`, de tipo `void`, que reciba como argumentos una matriz y su tamaño, y que calcule su norma. El resultado obtenido se debe retornar "por referencia" en una variable tipo `double` que también es recibida por la función como argumento.

Solución:

```
void norma_matriz(int n, int m, double A[][m], double* Norma) {
    double norma=0;
    int i,j;
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            norma+=A[i][j]*A[i][j];
        }
    }
    *Norma=sqrt(norma);
    return;
}
```

24. La siguiente función devuelve el determinante de una matriz 3×3 que es pasada por referencia como argumento de la propia función:

```
double calcula_determinante(double A[][3]) {
    double det;
    det=A[0][0]*A[1][1]*A[2][2]+A[0][1]*A[1][2]*A[2][0]+A[0][2]*A[1][0]*A[2][1]-
        A[0][0]*A[1][2]*A[2][1]-A[0][1]*A[1][0]*A[2][2]-A[0][2]*A[1][1]*A[2][0];
    return det;
}
```

Luego, en la función `main` podemos definir nuestra matriz de la siguiente forma (por ejemplo):

```
double A[3][3]={{1,-10,1},{-4,-5,6},{7,8,-9}};
```

y calcular el determinante mediante la instrucción

```
printf("El determinante es %g\n", calcula_determinante(A));
```

El objetivo de este ejercicio es reescribir la función anterior `calcula_determinante` para que haga el mismo cálculo pero debemos llamarla del siguiente modo:

```
printf("El determinante es %g\n", calcula_determinante(*A));
```

Solución:

```
double calcula_determinante(double *A) {
    double det;
    det=A[0]*A[4]*A[8]+A[1]*A[5]*A[6]+A[2]*A[3]*A[7]-
        A[0]*A[5]*A[7]-A[1]*A[3]*A[8]-A[2]*A[4]*A[6];
    return det;
}
```

25. La siguiente función devuelve el determinante de una matriz 3×3 que es pasada por referencia como argumento de la propia función:

```
double calcula_determinante(double A[][3]) {
    double det;
    det=A[0][0]*A[1][1]*A[2][2]+A[0][2]*A[1][0]*A[2][1]+A[0][1]*A[1][2]*A[2][0]-
        A[0][2]*A[1][1]*A[2][0]-A[0][0]*A[1][2]*A[2][1]-A[0][1]*A[1][0]*A[2][2];
    return det;
}
```

Luego, en la función `main` podemos definir nuestra matriz de la siguiente forma (por ejemplo):

```
double A[3][3]={{0,0,-3},{-1,-2,0},{0,-9,3}};
```

y calcular el determinante mediante la instrucción

```
printf("El determinante es %g\n", calcula_determinante(A));
```

El objetivo de este ejercicio es reescribir la función anterior `calcula_determinante` para que haga exactamente el mismo cálculo pero debemos llamarla del siguiente modo:

```
printf("El determinante es %g\n", calcula_determinante(A[0]));
```

Solución:

```
double calcula_determinante(double *A) {
    double det;
    det=A[0]*A[4]*A[8]+A[2]*A[3]*A[7]+A[1]*A[5]*A[6]-
        A[2]*A[4]*A[6]-A[0]*A[5]*A[7]-A[1]*A[3]*A[8];
    return det;
}
```

26. La siguiente función devuelve el determinante de una matriz 3×3 que es pasada por referencia como argumento de la propia función:

```
double calcula_determinante(double A[][3]) {
    double det;
    det=A[0][1]*A[1][2]*A[2][0]+A[0][0]*A[1][1]*A[2][2]+A[0][2]*A[1][0]*A[2][1]-
        A[0][1]*A[1][0]*A[2][2]-A[0][2]*A[1][1]*A[2][0]-A[0][0]*A[1][2]*A[2][1];
    return det;
}
```

Luego, en la función `main` podemos definir nuestra matriz de la siguiente forma (por ejemplo):

```
double A[3][3]={{4,3,1},{0,2,-1},{-3,-3,-3}};
```

y calcular el determinante mediante la instrucción

```
printf("El determinante es %g\n", calcula_determinante(A));
```

El objetivo de este ejercicio es reescribir la función anterior `calcula_determinante` para que haga exactamente el mismo cálculo pero debemos llamarla del siguiente modo:

```
printf("El determinante es %g\n", calcula_determinante(&A[0][0]));
```

Solución:

```
double calcula_determinante2(double *A) {
    double det;
    det=A[1]*A[5]*A[6]+A[0]*A[4]*A[8]+A[2]*A[3]*A[7]-
        A[1]*A[3]*A[8]-A[2]*A[4]*A[6]-A[0]*A[5]*A[7];
    return det;
}
```