Programación

Ejercicios Tema 6
Tipos de Datos Estructurados:
Vectores y Matrices

Autores:

M. Paz Sesmero Lorente
Paula de Toledo Heras
Fco. Javier Ordóñez Morales
Juan Gómez Romero
José A. Iglesias Martínez
José Luis Mira Peidro

SOLUCIONES

1. Escriba un programa que realice la lectura consecutiva, por teclado, de números enteros hasta que se introduzca el valor 0 (cero) o se hayan introducido 20 números, y los almacene en un vector, en orden inverso al de su introducción

```
#include <stdio.h>
#define DIM 20

int main(void)
{
   int v[DIM], num, i, j;
   /*Se solicita el valor que toman las componentes del
   vector y se almacenan a partir de la última posición (DIM-1)*/
   i=0;
   do
   {
      printf ("\nDeme un numero entero\n");
      scanf ("%i",&num);
      //Si el núm. leido no es cero se introduce en el vector
     if (num!=0){
         i++;
         v[DIM-i]=num;
      }
   }while (( num!=0)&&(i<DIM));

   //Se Muestran los elmentos del vector.
   if (i>0){
         printf("Los valores introducidos, mostrados en orden
inverso, son:\n");
         for (j=DIM-i; j<DIM; j++)
             printf("%i\t", v[j]);
   }
   else
        printf("No ha introducido ningún valor");
   printf("\n");
   return 0;
}
```

2. Escriba un programa en C que pida al usuario los valores de una tabla de elementos enteros de dimensión 4 filas por 3 columnas, y muestre en pantalla la mayor de las sumas de sus columnas.

```
#include <stdio.h>
#define FIL 3
#define COL 2
void leerMatriz(int m[FIL][COL]);
void mostrarMatriz(int m[FIL][COL]);
void sumarColumnas (int m[FIL][COL], int suma[COL]);
void calcularMaximo(int v[], int tam, int *max, int *c);

int main(void)
{
  int mat [FIL][COL];
  int suma[COL]; //Vector que almacena la suma de cada columna
  int max, col_max;
  int i,j;
```

```
  leerMatriz(mat);
  mostrarMatriz(mat);

  sumarColumnas (mat, suma);
  calcularMaximo(suma, COL, &max, &col_max);

  printf("\nLa suma de columnas mayor tiene el valor %i y
corresponde a la columna %i\n",max,col_max);
  system("PAUSE");
  return 0;
}

void leerMatriz(int m[FIL][COL]){
  /*Almacena en una matriz los datos introducidos por teclado*/
  int i, j;
  for (i=0; i<FIL; i++){
     for (j=0; j<COL; j++){
       printf("\n mat[%i][%i]=",i,j);
       scanf("%i", &m[i][j]);
     }
  }
  return;
}

void mostrarMatriz(int m[FIL][COL]){
  /*Muestra por pantallas los elementos de una matriz*/
  int i, j;
  for (i=0; i<FIL; i++){
     for (j=0; j<COL; j++){
       printf("%i\t",m[i][j]);
     }
     printf("\n");
  }
  return;
}

void sumarColumnas (int m[FIL][COL], int suma[COL]){
  /*Almacena en el vector suma el resultado de sumar los
    elemementos de cada columna:
    Parámetros:    int m[FIL][COL] Matriz cuyas columnas queremos
                   sumar
                   suma[COL] Vector en el que se almacena la suma
    Valor Retorno: Ninguno*/
  int i, j;
  for (j=0;j<COL;j++){
    suma[j]=0; //Inicializacion
    for (i=0; i<FIL; i++){
           suma[j]=suma[j]+m[i][j];
    }

  }
  return;
}
```

```
void calcularMaximo(int v[], int tam, int *max, int *c){
  /*Calcula el máximo de los elementos de un vector y la posición
que ocupa
    Parámetros:        int v[]  Vector de entrada
                       int tam  Dimensiones del vector
                       int *max Máx. de las componentes del vector
                       int *c   Componente con el valor máximo
    Valor de Retorno:  Ninguno*/

    int i;

    *max=v[0];  //Se asume que la 1ª componente tiene el max valor
    *c=0;
    for (i=1; i<tam; i++){
       if (v[i]>*max){
          *max=v[i];
          *c=i;
       }
    }
    return;

}
```

3. Escriba un programa que lea los valores de todos los elementos de dos vectores de enteros introducidos por el usuario, copie esos dos vectores en un tercero (uno a continuación del otro) y muestre sus valores por pantalla. Los tamaños de los dos vectores deben declararse como constantes.

```
#include <stdio.h>

# define TAM1 2
# define TAM2 3

void leerVector(int v[], int t);
void crearVector(int v1[], int t1, int v2[], int t2, int v3[]);
void mostrarVector(int v[], int t);

int main(void)
{
   int  v1[TAM1],v2[TAM2], v3[TAM1+TAM2];

   leerVector(v1, TAM1);
   leerVector(v2, TAM2);

   crearVector(v1, TAM1, v2, TAM2, v3);

   printf("El vector resultante es:\n");
   mostrarVector(v3, TAM1+TAM2);

   return 0;
}
```

```
void leerVector(int v[], int t){
   /*Almacena en un vector los datos introducidos por teclado
    Parámetros:    v[] Vector que queremos inicializar
                   t   Tamaño del vector
    Valor Retorno: Ninguno*/
   int i;

   printf("\n Introduzca los %i elementos del vector\n", t);
   for (i=0; i<t; i++){
       scanf("%d", &v[i]);
   }

   return;
}

void crearVector(int v1[], int t1, int v2[], int t2, int v3[]){
   /*Almacena en un vector los datos almacenados en otros 2
vectores
    Parámetros:    v1[], t1 Vector y tamaño del primer vector
                   v2[], t2 Vector y tamaño del segundo vector
                   v3[]     Vector resultado
    Valor Retorno: Ninguno*/

   int i;
   for (i=0;i<(t1+t2);i++)
   {
      if (i<t1)
        v3[i]=v1[i];
      else
        v3[i]=v2[i-t1];
   }
   return;
}

void mostrarVector(int v[], int t){
  /*Muestra por pantalla las componentes de un vector
    Parámetros:    v[] Vector que queremos inicializar
                   t   Tamaño del vector
    Valor Retorno: Ninguno*/
   int i;
   for (i=0;i<t;i++)
       printf("%i\t", v[i]);
   printf ("\n");
   return;
}
```

4. Escriba un programa que declare un array llamado tabla, cuadrado y de dos dimensiones, rellene todos sus elementos con el valor 1 y muestre dicho array. Luego el programa deberá poner a 0 todos los elementos de la diagonal principal y volver a mostrar el resultado.

```
#include <stdio.h>

#define TAM 3
//Al tratarse de una matriz cuadrada, los valors de FIL Y COL son
iguales

void leerMatriz(int m[TAM][TAM]);
void mostrarMatriz(int m[TAM][TAM]);
void cambiarDiagonal(int m[TAM][TAM]);
int main(void)
{
   int mat[TAM][TAM];
   int i,j;
   leerMatriz(mat);
   mostrarMatriz(mat);
   cambiarDiagonal(mat);
   mostrarMatriz(mat);
   system("PAUSE");
   return 0;
}

void leerMatriz(int m[TAM][TAM]){
  /*Almacena en una matriz los datos introducidos por teclado*/
  int i, j;
  for (i=0; i<TAM; i++){
     for (j=0; j<TAM; j++){
       printf("\n mat[%i][%i]=",i,j);
       scanf("%i", &m[i][j]);
     }
  }
  return;
}

void mostrarMatriz(int m[TAM][TAM]){
  /*Muestra por pantallas los elementos de una matriz*/
  int i, j;
  for (i=0; i<TAM; i++){
     for (j=0; j<TAM; j++){
       printf("%i\t",m[i][j]);
     }
     printf("\n");
  }
  printf("\n");
  return;
}

void cambiarDiagonal(int m[TAM][TAM]){
   /*Pone a 0 los elementos de la diagonal Principal*/
   int i;
   for (i=0;i<TAM;i++)
      m[i][i]=0;
      return ;
}
```

5. Escriba un programa que pida al usuario que introduzca por teclado el expediente de un alumno, sabiendo que éste se representa mediante una tabla de notas por asignatura (5) y convocatoria (máximo 6).El programa debe calcular y mostrar en pantalla la media ponderada de los aprobados, considerando que el peso de cada convocatoria (a representar mediante un vector) es: 1.25 para la 1ª convocatoria, 1.0 para la 2ª, 0.9 para la 3ª, 0.75 para la 4ª, 0.6 para la 5ª y 0.5 para la 6ª.
Nota. En la resolución de este ejercicios no es necesario usar funciones.

```
#include <stdio.h>
#define N_ASIG 5
#define MAX_CONV 6


int main(void)
{
  float notas[N_ASIG][MAX_CONV];
  float pesos[MAX_CONV]={1.25,1.0,0.9,0.75,0.6,0.5};
  float media=0;

  int i, j;
  int aprobado; //Variable booleana
  int expulsado; //Variable booleana

  expulsado=0;
  i=0;
  /*Se solicita la nota del alumno para cada asignatrua hasta
      a) completar el expediente ó
      b) el alumno resulta expulsado*/
  do{
      aprobado=0;
      j=0;
      /*Para cada asignatura se solicita la nota obtenida en cada
      convocatoria hasta que:
        a) El alumno aprueba
        b) Se supera el número máx de convocatorias permitido*/
      do {
          printf ("Introduzca la nota de la asignatura %d en la
conv. %d\n", i+1,j+1);
          scanf ("%f", &notas[i][j]);
          /*Si la asignatura se considera aprobada se actualiza
            el valor de la nota meda*/
          if (notas[i][j]>=5){
                aprobado=1;
                media=media+pesos[j]*notas[i][j];
          } //if
          else{
             j=j+1;
          }
      }while ((aprobado==0)&&(j<MAX_CONV));
      /*Si el alumno supera el número máximo de convocatorias
        permitido resulta expulsado. En caso contrario se solicita
        las calificaciones de otra asignatura*/
      if (j==MAX_CONV){
          expulsado=1;
      }
      else{
          i++;
      }
    }while ((!expulsado) && (i<N_ASIG));
```

```
        if (!expulsado){
            media=media/N_ASIG;
            printf ("La media del alumno es %f\n", media);
        }
        else
            printf("El alumno ha superado el maximo numero de
    convocatorias permitido\n");

        return 0;
    }
```

6. **La empresa de libros LIBROMIX tiene a la venta 6 títulos de libros del autor Carlos Ruiz Zafón:**
   - **"Las Luces de Septiembre" (denominado titulo 1),**
   - **"El Príncipe de la Niebla" (denominado titulo 2),**
   - **"El Palacio de Medianoche" (denominado titulo 3),**
   - **"Marina" (denominado titulo 4),**
   - **"La Sombra del Viento" (denominado titulo 5) y**
   - **"El Juego del Ángel" (denominado titulo 6).**

   Este año, en la feria del libro de Madrid, se venderán dichos libros en 3 casetas diferentes, denominadas Caseta nº 1, Caseta nº 2 y Caseta nº 3.
   Acabada la feria, se quieren informatizar los resultados de las ventas de cada libro en cada caseta. Así, se deberá realizar un programa en C que:

   a) **Almacene en una matriz la cantidad de ejemplares de cada uno de los 6 títulos que se han vendido en cada una de las 3 casetas. Dicha información se solicitará al usuario del siguiente modo:**

   > ***"Introduzca las ventas del título 1 en la Caseta nº 1:"*** _
   > ***"Introduzca las ventas del título 2 en la Caseta nº 1:"*** _
   > ….
   > ***"Introduzca las ventas del título 6 en la Caseta nº 3:"*** _

   b) **A partir de dicha matriz, se deberá almacenar en un vector el número de ejemplares totales vendidos (suma de las ventas en cada caseta) de cada uno de los 6 títulos.**

   c) **A partir de dicho vector, se deberá mostrar por pantalla el número de la caseta que más ejemplares haya vendido.**

   d) **Por último, se quiere obtener más información sobre el último libro del autor ("El Juego Del Ángel"). Para ello, se mostrará un listado con el número de ejemplares que de este libro se han vendido en cada caseta, indicar la caseta en la que se han vendido menos ejemplares y el número de ejemplares vendidos en esta última caseta.**

```c
#include <stdio.h>
//Declaramos las constantes que utilizaremos:
#define LIBROS  6  //LIBROS - FILAS
#define CASETAS 3  //CASETA - COLUMNAS


void leerMatriz(int m[LIBROS][CASETAS]);
void mostrarMatriz(int m[LIBROS][CASETAS]);
void sumarCASETASumnas (int m[LIBROS][CASETAS], int suma[CASETAS]);
void mostrarVector(int v[], int t);
void calcularMaximo(int v[], int tam, int *max, int *c);
void calcularMinimo(int v[], int tam, int *min, int *c);

int main(void){
  //Declaración de variables:
  int max, min;
  int casetaMasVentas, casetaMenosVentas;
  int matrizFeria[LIBROS][CASETAS];
  int ventasTotalesCaseta[CASETAS];

  leerMatriz(matrizFeria);
  //Imprimimos la matriz de ventas por caseta y libro.
  mostrarMatriz(matrizFeria);

  /*Calculamos la suma de los 6 LIBROS de cada caseta,
  y lo almacenamos en el vector ventasTotalesCaseta.*/
  sumarCASETASumnas(matrizFeria, ventasTotalesCaseta);

  /*Mostramos las ventas de cada caseta: Elementos del vector
    ventasTotalesCaseta*/
  printf("Ejemplares vendidos en cada caseta\n");
  mostrarVector(ventasTotalesCaseta, CASETAS);

  //Calculamos la caseta que más ejemplares ha vendido, es decir,
  //el valor máximo de ventasTotalesCaseta
  calcularMaximo(ventasTotalesCaseta, CASETAS, &max,
&casetaMasVentas);

  //Imprimimos la caseta con más ventas:
   printf("\nCASETA CON MAS VENTAS: %i \n\n", casetaMasVentas+1);

  printf("\nRanking de ventas de \"El Juego del Angel\": \n");
  /*Mostramos por pantalla el número de ejemplares de "El Juego del
   Angel vendidos en cada caseta y localizamos el mínimo.
  Se tiene en cuenta que los datos referidos a este libro se
  corresponden con la LIBROSa 5 de matrizFeria*/
  mostrarVector(matrizFeria[5], CASETAS);
  calcularMinimo(matrizFeria[5], CASETAS, &min,
&casetaMenosVentas);

  /*Mostramos por pantalla la caseta en la que menos ejemplares de
   "El Juego del Angel" se han vendido.*/
  printf("\nCASETA CON MENOS VENTAS DEL JUEGO DEL ANGEL: %i  CON %i
EJEMPLARES VENDIDOS \n\n", casetaMenosVentas+1, min);
  return 0;
}
```

```
void leerMatriz(int m[LIBROS][CASETAS]){
  /*Almacena en una matriz los datos introducidos por teclado*/
  int i, j;
  for (i=0; i<LIBROS; i++){
     for (j=0; j<CASETAS; j++){
       printf("Introduzca las ventas del titulo %i en la Caseta num
%i: ", i+1, j+1);
       scanf("%i", &m[i][j]);
     }
  }
  return;
}

void mostrarMatriz(int m[LIBROS][CASETAS]){
  /*Muestra por pantallas los elementos de una matriz*/
  int i, j;
  for (i=0; i<LIBROS; i++){
     for (j=0; j<CASETAS; j++){
       printf("%i\t",m[i][j]);
     }
     printf("\n");
  }
  return;
}
void sumarCASETASumnas (int m[LIBROS][CASETAS], int suma[CASETAS]){
  /*Almacena en el vector suma el resultado de sumar los
elememento s
     de cada CASETASumna:
     Parámetros:    int m[LIBROS][CASETAS] Matriz cuyas CASETASumnas
queremos sumar
                    suma[CASETAS] Vector en el que se almacena la
suma
     Valor Retorno: Ninguno*/
  int i, j;
  for (j=0;j<CASETAS;j++){
    suma[j]=0; //Inicializacion
    for (i=0; i<LIBROS; i++){
           suma[j]=suma[j]+m[i][j];
    }

  }
  return;
}

void mostrarVector(int v[], int t){
  /*Muestra por pantalla las componentes de un vector
     Parámetros:    v[] Vector que queremos inicializar
                    t   Tamaño del vector
     Valor Retorno: Ninguno*/
   int i;
   for (i=0;i<t;i++)
       printf("%i\t", v[i]);
   printf ("\n");
   return;
}
```

```
void calcularMaximo(int v[], int tam, int *max, int *c){
  /*Calcula el máximo de los elementos de un vector y su posición
    Parámetros:        int v[]  Vector de entrada
                       int tam  Dimensiones del vector
                       int *max Máx. de las componentes del vector
                       int *c   Componente con el valor máximo
    Valor de Retorno:  Ninguno*/

    int i;

    *max=v[0];  //Se asume que la 1ª componente tiene el max valor
    *c=0;
    for (i=1; i<tam; i++){
       if (v[i]>*max){
          *max=v[i];
          *c=i;
       }
    }
    return;

}

void calcularMinimo(int v[], int tam, int *min, int *c){
  /*Calcula el mín de los elementos de un vector su  posición
    Parámetros:        int v[]  Vector de entrada
                       int tam  Dimensiones del vector
                       int *min Mín de las componentes del vector
                       int *c   Componente con el valor mínimo
    Valor de Retorno:  Ninguno*/

    int i;

    *min=v[0];  //Se asume que la 1ª componente tiene el min valor
    *c=0;
    for (i=1; i<tam; i++){
       if (v[i]<*min){
          *min=v[i];
          *c=i;
       }
    }
    return;

}
```

7. Una empresa informática nos ha solicitado que desarrollemos una versión simplificada del juego de los barquitos. Para su diseño nos dan las siguientes especificaciones:

a. El tablero es una cuadrícula de 8x8 casillas.

b. Al inicio del juego se solicitarán las coordenadas de cada uno de los barquitos. Por simplicidad se admitirá que hay 8 barquitos, que cada uno está colocado en una fila, y que el número de casillas que ocupa cada barquito es 2. Por tanto, para situar un barquito en el tablero bastará con dar una coordenada admitiéndose que la segunda casilla ocupada por el barquito se corresponde con la casilla que está situada a la derecha de la indicada. Así, si las coordenadas dadas por el usuario

son (3,4) las casillas ocupadas por el barquito serán (3,4) y (3,5). Para posicionar un barquito en el tablero el sistema debe comprobar que las coordenadas introducidas son válidas.

c. **Una vez posicionados los 8 barquitos en el tablero comienza el juego. El juego constará de 8 rondas. En cada ronda el jugador dará las coordenadas de una casilla:**

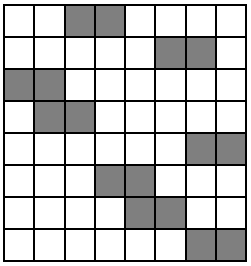
- **Si la casilla está ocupada un barquito, el barquito se considerará "hundido" y, por tanto, las 2 casillas ocupadas por este barquito serán casillas "libres" en las siguientes rondas. Además, al hundir un barquito, el sistema otorgará al usuario una ronda de juego adicional. Es decir, las rondas en las que el usuario hunde un barquito no serán descontadas del total de rondas disponibles.**
- **Si la casilla está libre, no es una casilla válida o está ocupada por un barquito ya hundido, se considerará que el usuario ha consumido una ronda.**

d. **El juego finaliza cuando se han hundido todos los barquitos o se han agotado todas las rondas.**

e. **Una vez finalizado el juego se indicará por pantalla si el jugador ha ganado o ha perdido. Si el jugador pierde la partida se mostrará por pantalla la distribución de barquitos inicial.**

Se pide desarrollar un programa en C que implemente el juego de los barquitos teniendo en cuenta las especificaciones pedidas.

Distribución inicial de una posible partida. Las celdas sombreadas indican la posición de los barquitos



```
#include <stdio.h>
#include <stdio.h>
#define DIM 8

void inicializarTablero (int tablero[DIM][DIM]);
void posicionarBarquitos (int tablero[DIM][DIM]);
void copiarTablero(int tablero[DIM][DIM], int copia[DIM][DIM]);
void mostrarTablero(int tablero[DIM][DIM]);
int main (void){

  int tablero[DIM][DIM];
  int copia [DIM][DIM];

  int i, j;
  int ronda;
  int hundidos;
```

```
  inicializarTablero(tablero);
  posicionarBarquitos(tablero);
  copiarTablero(tablero, copia);
  mostrarTablero(tablero);

   //Comienza el juego
  ronda=0;
  hundidos=0;
  do{
       printf ("Introduzca unas coordenadas ");
       scanf ("%d %d", &i, &j);
       //scanf ("%d", &j);
       if (tablero[i][j]==1){
           //Se pone a 0 la componente seleccionada y las dos
           //componentes adyacentes
           //Sólo sería necesario poner a 0 la que está a 1
           tablero[i][j]=0;
           if (j>0)
                tablero [i][j-1]=0;
           if (j<DIM-1)
               tablero [i][j+1]=0;
           //Se actualiza el valor de hundidos
           hundidos++;
           printf ("Ha hundido el barquito %d\n", hundidos);
       }
       else
         ronda++;
  } while ((hundidos<DIM)&&(ronda<DIM));

  if (hundidos==DIM)
     printf ("Enhorabuena. Ha ganado el juego\n");
  else{
     printf ("Lo siento. Ha perdido el juego\n");
     printf ("La situación inicial del tablero era\n");
     mostrarTablero(copia);
  }
  return 0;
}

void inicializarTablero (int tablero[DIM][DIM]){
    /*Pone a 0 las componentes de una matriz cuadrada
    Parámetros:    tablero[DIM][DIM]
    Valor Retorno: Ninguno*/
  int i, j;
  for (i=0; i<DIM; i++){
    for (j=0; j<DIM; j++){
       tablero[i][j]=0;
    }
  }
  return;
}
```

```
void posicionarBarquitos (int tablero[DIM][DIM]){
    /*Posiciona los barquitos en el tablero
    Parámetros:    tablero[DIM][DIM]
    Valor Retorno: Ninguno*/

  int j,k;
  int posValida;

  printf ("Introduzca la posición inicial de cada uno de los
barquitos\n");
  //Cada barquito se ubica en una fila. Se usa un bucle for
  for (k=0; k<DIM; k++){
     printf ("Barquito %d : ", k);
     do{
        posValida=1;
        //Se solicita la columna en la que estará ubicado el barco
scanf ("%d", &j);
        //Se verifica que la posición es válida. No se superan
        //las dimensiones del tablero. Hay que tener en cuenta que
        //cada barquito ocupa dos casillas
        if ((j<0) || (j>DIM-2)){
            posValida=0;
            printf ("Las coordenadas introducidas no son validas");
        }
     }while (posValida==0);
     //Tras comprobar que las coordenadas (columna) son válidas
     //se posiciona el barquito en el tablero.
     tablero[k][j]=1;
     tablero[k][j+1]=1;
  }
  return;
}

void copiarTablero(int tablero[DIM][DIM], int copia[DIM][DIM]){
   /*Copia la matriz tablero en la matriz copia
    Parámetros:    tablero[DIM][DIM]
                   copia[DIM][DIM]
    Valor Retorno: Ninguno*/
  int i, j;
  for (i=0; i<DIM; i++){
    for (j=0; j<DIM; j++)
        copia[i][j]=tablero[i][j];
  }
  return;
}

void mostrarTablero(int tablero[DIM][DIM]){
    /*Imprime una matriz cuadrada por pantalla
    Parámetros:    tablero[DIM][DIM]
    Valor Retorno: Ninguno*/
   int i, j;
   for (i=0; i<DIM; i++){
     for (j=0; j<DIM; j++){
       printf ("%d ", tablero[i][j]);
     }
     printf ("\n");
  }
  return;
}
```

8. Escriba las funciones necesarias para implementar las siguientes operaciones con matrices (representadas como arrays de dos dimensiones).
(NOTA: El tamaño de las matrices (número de filas y de columnas) será definido como una constante con la directiva #define).

a) Leer valores de matriz
Parámetros: Matriz `float m [FILAS][COLUMNAS]`
Devuelve: Nada
Acción: Lee valores para la matriz

b) Imprimir matrices
Parámetros: Matriz `float m[FILAS][COLUMNAS]`
Devuelve: Nada
Acción: Imprime m en pantalla

c) Obtener el valor máximo de los elementos de la matriz
Parámetros: Matriz `float m[FILAS][COLUMNAS]`
Devuelve: `double max`
Acción: Obtiene el valor máximo de la matriz

d) Obtener el valor mínimo de los elementos de la matriz
Parámetros: Matriz `float m[FILAS][COLUMNAS]`
Devuelve: `double min`
Acción: Obtiene el valor mínimo de la matriz

e) Sumar matrices
Entrada: Matrices
```
double m1[FILAS][COLUMNAS],
double m2[FILAS][COLUMNAS],
double r[FILAS][COLUMNAS]
```
Salida: Nada
Acción: Calcula r = m1 + m2

f) Restar matrices
Entrada: Matrices
```
double m1[FILAS][COLUMNAS],
double m2[FILAS][COLUMNAS],
double r[FILAS][COLUMNAS]
```
Salida: Nada
Acción: Calcula r = m1 - m2

g) Multiplicar matriz por valor escalar
Entrada: Matrices
```
double m[FILAS][COLUMNAS],
double r[FILAS][COLUMNAS]
```
Escalar `double x,`
Salida: Nada
Acción: Calcula r = x x m

h) Multiplicar matrices
Entrada: Matrices
```
double m1[FILAS][COLUMNAS],
double m2[COLUMNAS][FILAS],
double r[FILAS][FILAS]
```
Salida: Nada
Acción: Calcula r = m1 * m2

i) Obtener la traspuesta de una matriz
Entrada: Matrices `double m[FILAS][COLUMNAS],`
`double r[COLUMNAS][FILAS]`
Salida: Nada
Acción: Calcula r = traspuesta(m)

```
#include <stdio.h>

#define FIL 2
#define COL 3

void leerMatriz(float m[FIL][COL]);
void imprimirMatriz(float m[FIL][COL]);
float maxMatriz(float m[FIL][COL]);
float minMatriz(float m[FIL][COL]);
void sumarMatrices(float m1[FIL][COL], float m2[FIL][COL], float
r[FIL][COL]);
void restarMatrices(float m1[FIL][COL], float m2[FIL][COL], float
r[FIL][COL]);
void multiplicarMatrizEscalar(float m[FIL][COL], float x, float
r[FIL][COL]);
void multiplicarMatrices(float m1[FIL][COL], float m2[COL][FIL],
float r[FIL][FIL]);
void traspuesta(float m[FIL][COL], float r[COL][FIL]);

int main(void)
{
    float a[FIL][COL];
    float b[FIL][COL];
    float c[FIL][COL];
    float p[FIL][FIL];
    float t[COL][FIL];

    int i, j;

    /* Probar funciones */
    printf("\nLeer matriz a:\n");
    leerMatriz(a);
    printf("\nLeer matriz b:\n");
    leerMatriz(b);

    printf("\nMatriz a:\n");
    imprimirMatriz(a);
    printf("\nMatriz b:\n");
    imprimirMatriz(b);

    printf("\nMax a: %f\n", maxMatriz(a));
    printf("\nMin a: %f\n", minMatriz(a));

    sumarMatrices(a, b, c);
    printf("\nSuma a+b\n");
    imprimirMatriz(c);

    restarMatrices(a, b, c);
    printf("\nResta a-b\n");
    imprimirMatriz(c);
```

```
    multiplicarMatrizEscalar(a, 2.0, c);
    printf("\nMultiplicar a*2.0\n");
    imprimirMatriz(c);

    traspuesta(a, t);
    printf("\nTraspuesta\n");
    for(i=0; i<COL; i++) {
        for(j=0; j<FIL; j++)
            printf("%2.2f ", t[i][j]);
        printf("\n");
    }

    multiplicarMatrices(a, t, p);
    printf("\nMultiplicar a x b\n");
    for(i=0; i<FIL; i++) {
        for(j=0; j<FIL; j++)
            printf("%2.2f ", p[i][j]);
        printf("\n");
    }
}

void leerMatriz(float m[FIL][COL]) {
    int i, j;

    for(i=0; i<FIL; i++)
        for(j=0; j<COL; j++) {
            printf("Introduzca valor (%d, %d): ", i, j);
            scanf("%f", &m[i][j]);
        }

}

void imprimirMatriz(float m[FIL][COL]) {
    int i, j;

    for(i=0; i<FIL; i++) {
        for(j=0; j<COL; j++) {
            printf("%2.2f ", m[i][j]);
        }
        printf("\n");
    }
}

float maxMatriz(float m[FIL][COL]) {
    int i, j;
    float max = m[0][0];

    for(i=0; i<FIL; i++) {
        for(j=0; j<COL; j++) {
            if(m[i][j] > max)
                max = m[i][j];
        }
    }

    return max;
}
```

```
float minMatriz(float m[FIL][COL]) {
    int i, j;
    float min = m[0][0];

    for(i=0; i<FIL; i++) {
        for(j=0; j<COL; j++) {
            if(m[i][j] < min)
                min = m[i][j];
        }
    }

    return min;
}

void sumarMatrices(float m1[FIL][COL], float m2[FIL][COL], float
r[FIL][COL]) {
    int i, j;

    for(i=0; i<FIL; i++) {
        for(j=0; j<COL; j++) {
            r[i][j] = m1[i][j] + m2[i][j];
        }
    }
}

void restarMatrices(float m1[FIL][COL], float m2[FIL][COL], float
r[FIL][COL]) {
    int i, j;

    for(i=0; i<FIL; i++) {
        for(j=0; j<COL; j++) {
            r[i][j] = m1[i][j] - m2[i][j];
        }
    }
}

void multiplicarMatrizEscalar(float m[FIL][COL], float x, float
r[FIL][COL]) {
    int i, j;

    for(i=0; i<FIL; i++) {
        for(j=0; j<COL; j++) {
            r[i][j] = m[i][j] * x;
        }
    }
}

void multiplicarMatrices(float m1[FIL][COL], float m2[COL][FIL],
float r[FIL][FIL]) {
    int i, j, k;

    for(i=0; i<FIL; i++)
        for(j=0; j<FIL; j++) {
            r[i][j] = 0;
            for(k=0; k<COL; k++)
                r[i][j] += m1[i][k] * m2[k][j];
        }

}
```

```
void traspuesta(float m[FIL][COL], float r[COL][FIL]) {
    int i, j;

    for(i=0; i<FIL; i++) {
        for(j=0; j<COL; j++) {
            r[j][i] = m[i][j];
        }
    }
}
```

9. Escribir un programa que permita gestionar una máquina expendedora. El programa deberá pedir al usuario el precio del producto que va a comprar y el dinero que ha pagado y calcular la vuelta correspondiente. El programa debe mostrar cuántas monedas de cada tipo deben ser devueltas. Los tipos de monedas a considerar deben ser 1, 2, 5, 10, 20 y 50 céntimos, y 1 y 2 euros, y se asume que hay suficientes monedas de todos los tipos para devolver el cambio. Utilizar funciones para facilitar la legibilidad del programa principal.

```
#include <stdio.h>
#define NUM_MONEDAS 8

// Prototipos de las funciones
void calcularCambio (float cam_t,int aCam[NUM_MONEDAS],int
aVal[NUM_MONEDAS]);
void escribirCambio (int aCam[NUM_MONEDAS],char
aNom[NUM_MONEDAS][20]);

int main(void){
      float precio, pagado, cambio;
      int aCambio[NUM_MONEDAS];    /* vector en el que se guardará
        cuántas monedas de cada valor hay que devolver*/
    // Valor de cada moneda
      int aValores[NUM_MONEDAS]={200,100,50,20,10,5,2,1};
      // Nombre de cada moneda
      char aNombres[NUM_MONEDAS][20]={"2 euros", "1 euro", "50
centimos", "20 centimos","10 centimos","5 centimos","2 centimos","1
centimos"};

      // Leemos precio y dinero pagado
      // hasta que lo pagado sea mayor que el precio
      printf ("Escriba el precio del producto\n");
      scanf ("%f", &precio);

      printf ("Escriba el dinero pagado\n");
      scanf ("%f", &pagado);

      // si lo pagado es menor que el precio, volvemos a leer
      while (pagado<precio){
        printf ("Dinero insuficiente, escriba una cantidad mayor o
igual que el precio\n");
            scanf ("%f", &pagado);
    }

     //calculamos y mostramos el cambio en euros
      cambio = pagado - precio;
      printf ("Se van a devolver %.2f euros \n", cambio);
```

```
    // Llamamos a la función que calcula el cambio
     calcularCambio (cambio,aCambio,aValores);
     //Llamamos a la función que muestra los resultados
     escribirCambio(aCambio,aNombres);


       return 0;
}


void escribirCambio (int aCam[NUM_MONEDAS],char
aNom[NUM_MONEDAS][20]){
      // funcion que recibe como parametro el cambio a devolver aCam
      // y los nombres de cada moneda
      // y muestra el resultado por pantalla

       int i;
       //escribimos el resultado
       printf ("El cambio a devolver esta compuesto por:\n");
       for (i=0;i<NUM_MONEDAS; i++){
             if (aCam[i] != 0) {
                 printf ("%d  monedas de %s\n", aCam[i] , aNom[i]);
         }
       }
       return;
}

void calcularCambio (float cam_t,int aCam[NUM_MONEDAS],int
aVal[NUM_MONEDAS]){
// función que calcula el cambio de cam_t euros en monedas
// devuelve los resultados en el array de monedas mon
// necesita conocer los valores en céntimos de las monedas, que se
// pasan como parámetro en el array de valores val

    // variables locales
       int i, cam_cent;

       /* vamos dividiendo entre los valores de las monedas, de
         mayor a menor el dividendo de la división entera será el
         numero de monedas el resto de la división entera será el
         cambio que queda por devolver.
         como hay que trabajar con división entera, se pasa a
         céntimos*/
       cam_cent = cam_t * 100;

       for (i=0;i<NUM_MONEDAS;i++){
             aCam[i] = cam_cent/aVal[i];     // division entera
             cam_cent = cam_cent%aVal[i]; // resto
       }
       return;
}
```

10. Modifique el ejercicio anterior para que la vuelta se calcule suponiendo que hay disponibles para devolver sólo un número limitado de monedas de cada tipo (el numero de monedas disponible puede leerse por teclado o escribirse en el código)

```
#include <stdio.h>

#define NUM_MONEDAS 8

// Prototipos de las funciones
void escribirCambio (int aCam[NUM_MONEDAS],char
aNom[NUM_MONEDAS][20]);
int calcularCambioLimite (float cam_t,int aCam[NUM_MONEDAS],int
aVal[NUM_MONEDAS],int aMonDisponibles[NUM_MONEDAS]);

int main(void){
/* NOTA: Este programa es exactamente igual que el anterior,
  solo cambia la función CalcularCambio, que se sustituye por
  CalcularCambio_Limite. Estas dos funciones se diferencian en
  que la segunda tiene un argumento más, el número de
  monedas disponibles de cada tipo, que se tiene en cuenta
  para calcular la vuelta */


      float precio, pagado, cambio;
      int aCambio[NUM_MONEDAS];    /* vector en el que se guardará
                  cuantas monedas de cada valor hay que devolver*/
      // Valor de cada moneda
      int aValores[NUM_MONEDAS]={200,100,50,20,10,5,2,1};
      // Nombre de cada moneda
      char aNombres[NUM_MONEDAS][20]={"2 euros", "1 euro", "50
centimos", "20 centimos","10 centimos","5 centimos","2 centimos","1
centimos"};
    //Monedas disponibles de cada tipo
    int aMonDisponibles[NUM_MONEDAS]={2,2,2,2,2,2,2,2};
    int posible;  // Controla si es o no posible devolver el cambio

      //Monedas disponibles para dar cambio
      //Puede leerse por pantalla o escribirse en el código
      //Aquí se elige la primera opción

     // Leemos precio y dinero pagado
     // hasta que lo pagado sea mayor que el precio
      printf ("escriba el precio del producto\n");
      scanf ("%f", &precio);

      printf ("escriba el dinero pagado\n");
      scanf ("%f", &pagado);

      // si lo pagado es menor que el precio, volvemos a leer
      while (pagado<precio){
            printf ("dinero insuficiente, escriba una cantidad
mayor o igual que el precio\n");
            scanf ("%f", &pagado);
      }

     //calculamos y mostramos el cambio en euros
      cambio = pagado - precio;
      printf ("se van a devolver %f euros \n", cambio);
```

```
    // Llamamos a la función que calcula el cambio
     posible = calcularCambioLimite
(cambio,aCambio,aValores,aMonDisponibles);
     //Llamamos a la función que muestra los resultados
     if (posible==0){
        escribirCambio(aCambio,aNombres);
     }
     else{
        printf("No hay suficiente dinero para la vuelta\n");
     }

       system("pause");
       return 0;
}



int calcularCambioLimite (float cambio,int aCam[NUM_MONEDAS],int
aVal[NUM_MONEDAS],int aDisp[NUM_MONEDAS]){
/* función que calcula el cambio de cambio euros en monedas
   devuelve los resultados en el array de monedas mon
   necesita conocer los valores en centimos de las monedas,
   que se pasan como parametro en el vector de valores val
   tiene en cuenta cuantas monedas de cada tipo hay,
   que se pasa como parametro en el vector aDisp
   si no hay suficientes monedas para devolver todo,
   devuelve un -1. Si todo es correcto devuelve 0*/

     // variables locales
       int i, camCent,temp;


       /* vamos dividiendo entre los valores de las monedas, de
          mayor a menor el dividendo de la división entera será el
          numero de monedas el resto de la división entera será el
          cambio que queda por devolver como tengo que trabajar con
          división entera, paso a céntimos */
       camCent = cambio * 100;
       printf("la cantidad a devolver en centimos es %d\n",
camCent);

       for (i=0;i<NUM_MONEDAS;i++){
             temp = camCent/aVal[i];     // division entera
             //debería devolver temp unidades de esa moneda
             //pero tengo que ver si hay suficientes
             if (aDisp[i]>=temp){
            //si hay suficientes, devuelvo las necesarias y calculo
           // el nuevo disponible
                aCam[i] = temp;
                aDisp[i]=aDisp[i]-temp;
         }
         else{
             //si no hay suficiente disponible, devuelvo todas las
             //que haya de ese valor
             aCam[i] = aDisp[i];
             aDisp[i]=0;
         }
         // en los dos casos, la nueva cantidad a devolver es lo
         //que había menos lo que devuelvo de esa moneda
             camCent = camCent-(aCam[i]*aVal[i]);
```

```
        }
        //puede suceder que no haya suficientes monedas
        if (camCent !=0)
         camCent=-1;
     return(camCent);
}

void escribirCambio (int aCam[NUM_MONEDAS],char
aNom[NUM_MONEDAS][20]){
      // función que recibe como parámetro el cambio a devolver aCam
      // y los nombres de cada moneda
      // y muestra el resultado por pantalla

        int i;
        //escribimos el resultado
        printf ("el cambio a devolver esta compuesto por:\n");
        for (i=0;i<NUM_MONEDAS; i++){
              if (aCam[i] != 0){
                   printf ("%d  monedas de %s\n", aCam[i] , aNom[i]);
          }
        }
        return;
}
```

11. Escribir un programa en C que compruebe si una solución dada a un Sudoku es correcta sabiendo que:

- **a) Un tablero sudoku se compone de 81 casillas, 9 filas por 9 columnas. A su vez el tablero se subdivide en 9 subcuadrados de 9 casillas cada uno (3x3) NO SUPERPUESTOS (un número en una posición NO puede pertenecer a dos subcuadros).**
- **b) Se debe cumplimentar todas las casillas con un número comprendido entre el 1 y el 9.**
- **c) No puede repetirse ninguna cifra en la misma fila, ni en la misma columna ni en el mismo subcuadrado.**

3	8	1	9	7	6	5	4	2
2	4	7	5	3	8	1	9	6
5	6	9	2	1	4	8	7	3
6	7	4	8	5	2	3	1	9
1	3	5	7	4	9	6	2	8
9	2	8	1	6	3	7	5	4
4	1	2	6	8	5	9	3	7
7	9	6	3	2	1	4	8	5
8	5	3	4	9	7	2	6	1

Ejemplo de una solución correcta

```
/*SUDOKU
Un sudoku es correcto si en cada casilla hay un número del 1 a. 9
y:
      a. No se repite ningún número en ninguna fila.
      b. No se repite ningún número en ninguna columna.
      c. No se repite ningún número en ningun cuadrante.
Por tanto, para comprobar si el sudoku es correcto se deben
analizar todas
las filas, todas las columnas y todos los cuadrantes comprobando
```

```
que
en cada uno de ellos aparecen todos los números del 1 al 9. Si
algún número
no aparece significa que otro número está repetido y, por tanto, el
sudoku
es incorrecto.

Para comprobar filas, columnas y cuadrantes se usa un vector
auxiliar de 9
casillas en el que se volcará cada una de las filas, columnas o
cudrantes.
*/
#include <stdio.h>
void mostrar(int s[][9]);
int comprobarVector(int v[]);

int main(void)
{
 //Ejemplos de sudoku

 //Correcto
 /*int
sudoku[9][9]={{3,8,1,9,7,6,5,4,2},{2,4,7,5,3,8,1,9,6},{5,6,9,2,1,4,
8,7,3},

{6,7,4,8,5,2,3,1,9},{1,3,5,7,4,9,6,2,8},{9,2,8,1,6,3,7,5,4},

{4,1,2,6,8,5,9,3,7},{7,9,6,3,2,1,4,8,5},{8,5,3,4,9,7,2,6,1}}; */
 //Incorrecto: En la segunda fila aparece dos veces el número 3
 /*int
sudoku[9][9]={{3,8,1,9,7,6,5,4,2},{3,4,7,5,3,8,1,9,6},{5,6,9,2,1,4,
8,7,3},

{6,7,4,8,5,2,3,1,9},{1,3,5,7,4,9,6,2,8},{9,2,8,1,6,3,7,5,4},

{4,1,2,6,8,5,9,3,7},{7,9,6,3,2,1,4,8,5},{8,5,3,4,9,7,2,6,1}};*/

 //Incorrecto: En la primera columna aparecen números repetidos
 /*int
sudoku[9][9]={{1,2,3,4,5,6,7,8,9},{1,2,3,4,5,6,7,8,9},{1,2,3,4,5,6,
7,8,9},

{1,2,3,4,5,6,7,8,9},{1,2,3,4,5,6,7,8,9},{1,2,3,4,5,6,7,8,9},

{1,2,3,4,5,6,7,8,9},{1,2,3,4,5,6,7,8,9},{1,2,3,4,5,6,7,8,9}};*/

 //Incorrecto: En el primer cuadrante aparecen números repetidos
  int
sudoku[9][9]={{1,2,3,4,5,6,7,8,9},{2,3,4,5,6,7,8,9,1},{3,4,5,6,7,8,
9,1,2},

{4,5,6,7,8,9,1,2,3},{5,6,7,8,9,1,2,3,4},{6,7,8,9,1,2,3,4,5},

{7,8,9,1,2,3,4,5,6},{8,9,1,2,3,4,5,6,7},{9,1,2,3,4,5,6,7,8}};

  int f,c,kf,kc;    //Indices auxiliares para recorrer filas y
columnas
  int j;
  int vector[9];    //Vector en el que se vuelca cada fila, columna
o cuadrante
```

```
    int correcto=0;    //Variable que indica si, hasta el momento
dado, el sudoku
                       // es correcto

    mostrar(sudoku);   //Se muestra la solución propuesta.

    //Analisis de las filas. Los elementos de cada fila se almacenan
    //en el vector auxiliar. Al menos es necesario analizar la
    //primera fila. Se usa por tanto una estructura do while
    f=0;
    do{
        for (c=0; c<9; c++)
            vector[c]=sudoku[f][c];
        correcto=comprobarVector(vector);  /*Se comprueba si el
            vector contiene todos los números del 1 al 9*/
        f++;
    }while ((correcto==1)&&(f<9));


    //Análisis de las columnas.
    //Solo se analizan las columnas si todas las filas son correctas.
    //Se usará por tanto una estructura while

    c=0;
    while((correcto==1)&&(c<9)){
        for (f=0; f<9; f++)
            vector[f]=sudoku[c][f];
        correcto=comprobarVector(vector);
        c++;
    };


    /*Análisis de los cuadrantes. En este punto es fundamental
      determinar las casillas en las que empieza cada cuadrante :
        (0,0), (0,3), (0,6)
        (3,0), (3,3), (3,6)
        (6,0), (6,3), (6,6)*/

    kf=0;
    kc=0;
    //kf y kc son los índices de las casillas
    f=0;
    c=0;
    //f y c Determinan la casilla de inicio de cada cuadrante.
    while((correcto==1)&&(f<9)&&(c<9)){
        j=0;
        for (kf=f; kf<f+3; kf++)
           for(kc=c; kc<c+3; kc++){
              vector[j]=sudoku[kf][kc];
              j++;
        }
        correcto=comprobarVector(vector);
        //Se actualizan los índices:
        if (c<6)
           c=c+3;
        else{
           c=0;
           f=f+3;
        }
    };
```

```
  if (correcto==1)
      printf("El sudoku es correcto\n");
  else
      printf("El sudoku es incorrecto. Revise la solución\n");
  system("PAUSE");
  return (0);
};
void mostrar(int s[][9])
{
  int i, j;
  for (i=0; i<9; i++){
    for (j=0; j<9; j++)
       printf("%i ",s[i][j]);
    printf("\n");
  }
  return;
};

int comprobarVector(int v[])
{
  /*Comprueba que el vector contiene todos los números del 1 al 9.
    Hay distintas formas de hacerlo. Una de ellas es comprobar
    que todos los números están en el vector. Si un determinado
    número no está, comprobarVector=false (0). Se podría hacer
    también ordenando el vector y comprobando que el mismo
    coincide con {1,2,3,4,5,6,7,8,9}.
  */
  int encontrado;  // Determina si un número está en el vector

  int i,j;


  i=1;       //Primer número a buscar
  do{
    encontrado=0;
    j=0;     //Primera posición a analizar
    do{
      if (v[j]==i)
        encontrado=1;
      else
         j++;
    }while ((encontrado==0)&&(j<9));  /*La búsqueda continua
                         mientras el número no se ha encontrado
                         y no se han analizado todas las casillas
                         (j<9).  */
    if (encontrado)
         i++;     //Se actualiza el número a buscar
  }while((encontrado==1)&&(i<=9));   /*La búsqueda se detiene
                         cuando un número no se ha encontrado*/
  return(encontrado);
}
```