1. Escriba un programa en C que reciba por línea de comandos el número de filas de una matriz, el número columnas y el nombre de un fichero. Valide que los números estén comprendidos en el intervalo [1,10] y a continuación pida el valor de cada elemento de la matriz al usuario por teclado (números reales). Una vez leídos todos los valores, la matriz se guardará en el fichero cuyo nombre se pasó por línea de comandos, separando las columnas con tabuladores y las filas por retornos de carro. Si hay algún error al abrir el fichero, se mostrará un error al usuario. Todas las salidas de errores han de escribirse en la salida estándar de error.

   Valide que el número de filas no puedan ser valores no numéricos. Por ejemplo la entrada:

   ```
   ./a.out a2 3b fichero.txt
   ```

   deberá mostrar un error.

   Use una matriz cuya memoria debe alojarse de manera dinámica.

   Un ejemplo de uso sería:

   ```
   ./a.out 2 3 fichero.txt
   v[0,0]: 2.4
   v[0,1]: 8.4
   v[0,2]: 17.9
   v[1,0]: -5.2
   v[1,1]: 4.97
   v[1,2]: -2.1
   Matriz guardada en fichero.txt!
   ```

2. Repita el ejercicio 1 pero con 4 parámetros de entrada:

   a) Número de filas de la matriz `A`.

   b) Número de columnas de la matriz `A` y número de filas de la matriz `B`.

   c) Número de columnas de la matriz `B`.

   d) Nombre del fichero de salida.

   Valide la entrada, del mismo modo que el ejercicio 1, y pida los valores de ambas matices por teclado. A continuación, multiplique las matrices, muestre el resultado por pantalla y guárdelo en el fichero de salida.

   Todas las matrices deben ser memoria alojada de manera dinámica y la multiplicación de matrices debe hacerse en una función.

Los siguientes ejercicios permiten ver ejemplos de los distintos conceptos presentados en las sesiones de prácticas. Como trabajo autónomo, codifique, entienda y responda a las preguntas que se formulan en alguno de ellos.

3. Reserva dinámica de memoria

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   printf("El tamano de un entero en memoria es de %lu bytes.\n",sizeof(int));
   printf("El tamano de un long en memoria es de %lu bytes.\n",sizeof(long));
```

```c

   int* a;
   //Esta variable no apunta a nada. como consigo memoria para guardar valores?
   a = (int *)malloc(sizeof(int));
   if (a == NULL)
   {
      printf("Error asignando memoria\n");
      exit(-1);
   }

   *a=5;
   printf("La variable a es un puntero a entero que apunta a la direccion %p\n",a);
   printf("El entero almacenado en la direccion %p es %d\n",a,*a);
   free(a);
   return 0;
}
```

¿Cuál es la diferencia entre `return` y `exit`? ¿Por qué cree que es útil el uso de la macro `sizeof`? ¿Qué devuelve y de qué tipo `sizeof`?

4. Rereserva dinámica de memoria: arrays

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   int num;
   float* v;
   printf("Introduzca un numero entero: ");
   scanf("%d",&num);

   if (num >0)
   {
      if ((v = (float *)malloc(num*sizeof(float)))==NULL)
      {
         printf("Error asignando memoria\n");
         exit(-1);
      }
      printf("Se ha reservado memoria a partir de la direccion: %p\n",v);

      for (int i=0; i<num; i++)
      {
         printf("Introduzca el valor de v[%d]: ",i);
         scanf("%f",&v[i]);
      }
      for (int i=0; i<num; i++)
      {
         printf("Valor de v[%d]: %g en la direccion %p\n",i,v[i],&v[i]);
      }
      free(v);
   }
  return 0;
}
```

¿Por qué en la línea `scanf("%f",&v[i]);` se utiliza el operador `&`? ¿Cual es la diferencia

entre las expresiones `v`, `v[i]` y `&v[i]` y de que tipo es cada una en el contexto de este ejercicio? ¿`v` y `&v[0]` es lo mismo en el contexto de este ejercicio?
Estudie el siguiente programa en C donde se utiliza la aritmética de punteros ¿Es equivalente al anterior?

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   int num;
   float *v, *p;
   printf("Introduzca un numero entero: ");
   scanf("%d",&num);

   if (num >0)
   {
      if ((v = (float *)malloc(num*sizeof(float))) == NULL) {
         fprintf(stderr, "Error, no hay memoria\n");
         exit(-1);
      };
      p=v;
      printf("Se ha reservado memoria a partir de la direccion: %p\n",v);

      for (int i=0; i<num; i++)
      {
         printf("Introduzca el valor de v[%d]: ",i);
         scanf("%f",v++);
      }
      v=p;
      for (int i=0; i<num; i++)
      {
         printf("Valor de v[%d]: %g en la direccion %p\n",i,*v, v);
         v++;
      }
      free(v);
   }
  return 0;
}
```

¿Por que se utiliza `fprintf` en lugar de `printf`? ¿A qué se refiere ese comando con `stderr`? ¿tiene algo que ver con los comandos y las redirecciones que se han estudiado en la parte de UNIX?

5. Rereserva dinámica de memoria: arrays de structs.
Observe el siguiente código en C y asegúrese de que lo entiende.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct bloque {
   int i;
   char c;
} BLOQUE;

int main(void)
{
   int num;
```

```
12    BLOQUE *v, *p;
13    printf("Introduzca un numero entero: ");
14    scanf("%d",&num);
15
16    if (num >0)
17    {
18       if ((v = (BLOQUE *)malloc(num*sizeof(BLOQUE))) == NULL) {
19          fprintf(stderr, "Error, no hay memoria\n");
20          exit(-1);
21       };
22       p=v;
23       printf("Se ha reservado memoria a partir de la direccion: %p\n",v);
24
25       for (int i=0; i<num; i++)
26       {
27          printf("Introduzca el valor de v[%d]->i: ",i);
28          scanf("%i",&v->i);
29          printf("Introduzca el valor de v[%d]->c: ",i);
30          scanf("%*c%c",&v->c);
31          v++;
32       }
33       v=p;
34       for (int i=0; i<num; i++)
35       {
36          printf("Valor de v[%d]->i: %i v[%d]->c %c en la direccion %p\n",i
    ,v->i, i, v->c,v);
37          v++;
38       }
39       free(v);
40    }
41   return 0;
42 }
```

¿Qué es `BLOQUE` en este programa? ¿Podría saberse cuánto ocupa en memoria? ¿Cómo? Traduzca el código para usar el operador `[]` en lugar de la aritmética de punteros. Es decir, que no sea necesario usar `v++`, por ejemplo.

6. Paso de vectores a funciones.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define TAM1 5
5  #define TAM2 7
6
7  float sumaVector(float v[], int tam);
8
9  int main()
10 {
11    float v[TAM1] = {1.0, 2.0, 3.0, 4.0, 5.0};
12    float* v2;
13
14    printf("La suma del vector v es %g\n",sumaVector(v,TAM1));
15
16    v2 = (float *)malloc(TAM2*sizeof(float));
17    v2[0] = 1.1;
18    v2[1] = 2.1;
19    v2[2] = 3.1;
```

```
20    v2[3] = 4.1;
21    v2[4] = 5.1;
22    v2[5] = 6.1;
23    v2[6] = 7.1;
24
25    printf("La suma del vector v es %g\n",sumaVector(v2,TAM2));
26    free(v2);
27    return 0;
28 }
29
30 float sumaVector(float v[], int tam)
31 {
32    float suma = 0.0;
33    for(int i=0; i<tam; i++)
34    {
35       suma += v[i];
36    }
37    return suma;
38 }
```

¿Por qué el programador ha decidido pasar el tamaño del vector como un parámetro más? ¿Identifica alguna diferencia en trabajar con vectores declarados de manera dinámica o estática? ¿Se debería hacer alguna comprobación después de asignar memoria con `malloc`? ¿Se le ocurre otra forma de declarar la función `sumaVector`?
¿Por qué se ha añadido el fichero de cabecera `stdlib.h`?

7. Parámetros de la función `main`.

```
1  #include <stdio.h>
2
3  #define TAM 200
4
5  int main(int argc, char* argv[])
6  {
7     int num;
8     char cadena[TAM];
9     printf("El numero de parametros pasados es %d\n",argc);
10    printf("Los parametros son:\n");
11    for (int i=0;i<argc; i++)
12    {
13       printf("\tParametro %d: %s.\t",i,argv[i]);
14       if (sscanf(argv[i],"%d",&num)!=0)
15       {
16          printf("Parametro entero con valor %d.\n",num);
17       }
18       else
19       {
20          printf("Parametro no entero\n");
21       }
22    }
23
24    return 0;
25 }
```

Responda a las siguientes preguntas:
¿Qué es `argv`? ¿un array de enteros, de caracteres o de punteros a cadenas de caracteres?
¿Por qué se ha hecho una reserva de 200 caracteres para la variable `cadena`? ¿Se le ocurre

otra manera de hacer una reserva de memoria para la variable `cadena`?
Estudie el uso de la función `sscanf` ¿Se le ocurre otra forma de transformar una cadena de caracteres en un entero? ¿Qué devuelve la función `sscanf`?
En la línea `printf(''\tParametro %d: %s.\t'',i,argv[i]);` cuando se pasa el argumento `argv[i]` ¿de qué tipo de datos es?
Si quisiera leer una cadena de caracteres (sin espacios) con la función `scanf` y que el resultado de la lectura se almacenara en la variable `cadena` (declarada como `char candena[200]`) ¿cómo lo escribía `scanf(''%s'',&cadena)` o `scanf(''%s'',cadena)`? ¿Por qué?
¿Cuál es el primer argumento del `main`?
Pruebe a ejecutar el programa con los siguientes argumentos y medite el resultado:

```
.\porgramaC 1 1a a1 abc 12.45
```

8. Estructuras o `structs`.
   Escriba y analice el siguiente código

```c
/* Ejemplos de structuras*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main (void) {
/* definicion de estructura con nombre */
   struct fecha {
      int dia;
      int mes;
      int anio;
   };

   struct persona {
      char nombre[25];
      char direccion[50];
      char *cadena;
      long codigo_postal;
      long seguridad_social;
      double salario;
      struct fecha cumpleanios;
      struct fecha contrato;
   };
/* definicion de variables tipo estructura */
   struct persona p1, p2, *p;

/* Inicializacion */
   struct persona p3 = {"perico", "mi direccion de casa", NULL, 47001,
   123456789, 35500,  {12, 01, 1960}, {15, 1, 2020}    };

/* Definicion de la estrutura y variables. se puede no poner nombre */
   struct {
      int i;
      char linea[20];
   } dato1, datos2;

/* ERROR COMUN se pueden igualar las estructuras como las variables de
   tipo basico pero si la estructura tiene punteros NO!!! se debe hacer
   por partes:
   struct ej {
```

```
38        char* cadena;
39    };
40    dest_struct = source_struct;
41    dest_struct.cadena = malloc(strlen(source_struct.cadena) + 1);
42    strcpy(dest_struct.cadena, source_struct.cadena);
43 */
44
45    if ((p3.cadena = (char *) malloc(sizeof(char)*strlen("hola"))) == NULL)
       {
46        fprintf(stderr, "Error al reservar memoria\n");
47        exit(-1);
48    }
49    strcpy(p3.cadena , "1234");
50    p1 = p3;
51    // Puntero a estructura
52    p = &p1;
53    //Primer printf
54    printf("Datos de persona P1 Puntero:\n \tNombre:%s\n \tDireccion:%s\n \
      tCadena:%s\n \tCodigo Postal:%ld\n \tSeguridad Social:%ld\n \tSalario:
      %f\n \tFecha Cumpleanios:%d/%d/%d\n \tFecha Contrato: %d/%d/%d\n",
55    p->nombre, p->direccion, p1.cadena, p->codigo_postal, p->
      seguridad_social, p->salario, p->cumpleanios.dia, p->cumpleanios.mes, p
      ->cumpleanios.anio, p->contrato.dia, p->contrato.mes, p->contrato.anio)
      ;
56    printf("Datos de persona P3:\n \tNombre:%s\n \tDireccion:%s\n \tCadena
      :%s\n \tCodigo Postal:%ld\n \tSeguridad Social:%ld\n \tSalario: %f\n \
      tFecha Cumpleanios:%d/%d/%d\n \tFecha Contrato: %d/%d/%d\n",
57    p3.nombre, p3.direccion, p3.cadena, p3.codigo_postal, p3.
      seguridad_social, p3.salario, p3.cumpleanios.dia, p3.cumpleanios.mes,
      p3.cumpleanios.anio, p3.contrato.dia, p3.contrato.mes, p3.contrato.anio
      );
58
59     // Cambiamos p1
60     printf("======== MODIFICAMOS DATOS de P1 =========\n");
61     p1.salario = 12000;
62     strcpy(p->nombre, "Manolo");
63     p->codigo_postal = 47014;
64     p->cumpleanios.dia = 20;
65     p1.cumpleanios.mes =12;
66     strcpy(p1.cadena, "hola");
67     //Tercer printf
68     printf("Datos de persona P1 Puntero:\n \tNombre:%s\n \tDireccion:%s\n
      \tCadena:%s\n \tCodigo Postal:%ld\n \tSeguridad Social:%ld\n \tSalario:
       %f\n \tFecha Cumpleanios:%d/%d/%d\n \tFecha Contrato: %d/%d/%d\n",
69     p->nombre, p->direccion, p->cadena, p->codigo_postal, p->
      seguridad_social, p->salario, p->cumpleanios.dia, p->cumpleanios.mes, p
      ->cumpleanios.anio, p->contrato.dia, p->contrato.mes, p->contrato.anio)
      ;
70     //Cuarto printf
71     printf("Datos de persona P3:\n \tNombre:%s\n \tDireccion:%s\n \tCadena
      :%s\n \tCodigo Postal:%ld\n \tSeguridad Social:%ld\n \tSalario: %f\n \
      tFecha Cumpleanios:%d/%d/%d\n \tFecha Contrato: %d/%d/%d\n",
72     p3.nombre, p3.direccion, p3.cadena, p3.codigo_postal, p3.
      seguridad_social, p3.salario, p3.cumpleanios.dia, p3.cumpleanios.mes,
      p3.cumpleanios.anio, p3.contrato.dia, p3.contrato.mes, p3.contrato.anio
      );
73     printf("CUIDADO p1.cadena es la misma variable que p3.cadena. NO
      OCURRE IGUAL CON LAS VARIABLES TIPO\n");
```

```
74
75      free(p1.cadena);
76      return 0;
77  }
```

Conteste a las siguientes preguntas:
¿Qué es un `struct` o estructura? ¿Lo ha estudiado en algún otro lenguaje de programación?
¿De qué tipo es la variable `p`? ¿Qué almacenará?
¿Se podría inicializar la variable `p3` con el operador `.`?
¿Para que sirven las funciones `strlen` y `strcpy`? ¿Qué devuelven y qué parámetros reciben?
¿Qué fichero de cabecera hay que incluir para usarlas?
¿Está seguro que el primer `printf` escribe los valores de `p1`? ¿Por qué? ¿Por qué se usa el operador `->`?
El miembro `cadena` de `p3` se ha reservado e inicializado con los valores ``1234'', luego al miembro `cadena` de `p1` se le ha asignado el valor ``hola''. ¿Sabría explicar por qué en el tercer y cuarto `printf` muestran que el contenido del miembro `cadena` es ``hola''? Trate de representar gráficamente el contenido de cada estructura y miembro.

9. Estudie el siguiente programa escrito en `C`:

```c
/* Ejemplo de typedef para definir bool en C */
#include <stdio.h>

#define true 1
#define false 0

typedef unsigned short char BOOL;

int main(void) {
   BOOL dato = true;

   if (dato == true)
      printf("dato es verdad\n");
   dato = false;
   if (dato == false)
      printf("dato ahora es false\n");
}
```

¿De qué tipo son realmente las variables declaradas como `BOOL`? ¿y que valore representa las constantes `true` y `false`?
Mediante el uso de `typedef` defina un tipo de datos que permita almacenar puntos en el espacio (x_1, x_2, x_3) con una descripción de hasta 50 caracteres. Es decir el valor de x_1, x_2, x_3 y la descripción.

10. Lea y comprenda el siguiente código escrito en `C`.

```c
#include<stdio.h>
#include<stdlib.h>

#define MAXLENGHT 255
#define NPAR      1

typedef struct dato {
  unsigned long int f1;
  unsigned long int f2;
} DATOS;
```

```
11
12 int main(void)
13 {
14
15   FILE *f_in;
16   int ndata=0;
17   char data[MAXLENGHT];
18   DATOS d;
19
20   if (argc != NPAR+1) {
21      fprintf(stderr, "Error debe poner el nombre del archivo\n leef
     Nombre_de_archivo\n");
22      exit(-1);
23   }
24   if ((f_in=fopen(argv[NPAR], "r")) == NULL) {
25      fprintf(stderr, "ERROR: Fichero de entrada no existe o no se puede
     leer.\n");
26      exit(-1);
27   }
28   else
29   {
30      // Leo las dos primeras lineas.
31      //fscanf(f_in, "%s", data);
32      if (fgets(data, MAXLENGHT, f_in) == NULL) {
33         fprintf(stderr, "ERROR: No puedo leer la primera linea de %s\n",
     argv[NPAR]);
34         exit(-1);
35      }
36      printf("Primera linea: %s\n", data);
37      if (fgets(data, MAXLENGHT, f_in) == NULL) {
38         fprintf(stderr, "ERROR: No puedo leer la segunda linea de %s\n",
     argv[NPAR]);
39         exit(-1);
40      }
41      printf("Segunda linea: %s\n", data);
42      // Contamos el numero de lineas del archivo.
43      while (fgets(data, MAXLENGHT, f_in)) {
44         sscanf(data, "%lu %lu", &d.f1, &d.f2);
45         printf("Linea %d: %lu %lu\n", ndata, d.f1, d.f2);
46         ndata++;
47      }
48      fclose(f_in);
49      printf("El numero de datos es: %d\n", ndata);
50   }
51   return (0);
52 }
```

¿Qué estructura parece tener el fichero que se está leyendo? Explique el uso de `sscanf` en este ejercicio. ¿De que tipo es la variable `f_in`?
¿Por qué en la línea 33 se usa `argv[NPAR]`? ¿De qué tipo es? ¿Qué mostrará el `printf`?