EJERCICIOS DE LA ASIGNATURA "PROGRAMACIÓN" ORDENADOS POR TEMAS

Escuela Politécnica Superior
Universidad de Alcalá

GRADO EN INGENIERIA EN TECNOLOGIAS DE LA TELECOMUNICACION (GITT)
GRADO EN INGENIERIA EN SISTEMAS DE TELECOMUNICACION (GIST)
GRADO EN INGENIERIA ELECTRONICA DE COMUNICACIONES (GIEC)
GRADO EN INGENIERIA TELEMATICA (GIT)

Fecha Actualización: 10/07/2019

PUNTEROS A ARRAYS Y A CADENAS DE CARACTERES

1. Utilizando exclusivamente estas declaraciones, realizar un programa que pida una palabra e imprima el número de vocales que contiene (tanto mayúsculas como minúsculas).
char pal[20]; char *pun=pal; int con=0;

Solución:

```
main()
{
  char pal[20]; char *pun=pal; int con=0;
  printf("Dame una palabra: "); gets(pal);
  while (*pun!=0)
  { if (*pun=='a' || *pun=='A' || *pun=='e' || *pun=='E' || *pun=='i' || *pun=='I' || *pun=='o' || *pun=='O'
|| *pun=='u' || *pun=='U')
      con++;
    pun++;
  }
  printf("\nEl numero de vocales es: %d\n", con);
}
```

---

2. Hacer una función con el siguiente prototipo: **char *fun(char *cad);**
donde el argumento **cad** es la dirección de comienzo de una cadena de caracteres que contiene una línea de texto en letras minúsculas. La función **fun()** debe transformar dicho texto en la misma cadena **cad** para que sea igual pero sustituyendo las letras minúscuals por mayúsculas. La función además debe devolver la dirección de comienzo de la cadena transformada.

Solución:

```
char *fun(char *cad)
{
  char *ini=cad;
  while(*cad)
  { if (*cad>='a' && *cad<="z") *cad+=('A'-'a');
    cad++;
  }
  return ini;
}
```

---

3. Escribir una función **void fun(char*s1, char*s2, char car)** que recibe dos cadenas de caracteres **s1** y **s2** y un carácter **car** desde el programa principal. La función deberá copiar los caracteres de **s1** en **s2** hasta llegar al final de la cadena, o hasta encontrar un carácter igual al de la variable **car.**
Ejemplo:
Si s1="Examen de programación" y car='v', entonces s2 será "Examen de programación"
Si s1="Examen de programación" y car='p', entonces s2 será "Examen de "

Solución:

```
void fun (char *s1, char *s2, char car)
{
  while (*s1!=car && *s1!='\0')
  {
    *s2=*s1; s2++; s1++;        // o bien:    *s2++ = *s1++;
  }
  *s2='\0';
}
```

O bien:

```
void fun (char *s1, char *s2, char car)
{
  int i=0;
  while (s1[i]!=car && s1[i]!='\0')
  {
    s2[i]=s1[i];   i++;
  }
  s2[i]='\0';
}
```

---

4. Escribir un programa que lea desde el teclado una frase compuesta por varias palabras en minúsculas (máximo 200 caracteres) y, con la ayuda de un puntero, la transforma para poner en mayúsculas solo la primera letra de cada palabra (si es una minúscula). Se entiende que la letra inicial es la que no es espacio en blanco pero está precedida de un espacio, o bien es la primera letra de toda la frase.

Solución:

```
#include <stdio.h>
main( )
{ char cad[200], *p=cad;  //el puntero p comienza desde el principio de cad
  printf("Teclea una frase: "); gets(cad);
  while (*p!=0)   //mientras p no alcance el character \0 de final de cadena
  { if (p==cad || (*p!=' ' && *(p-1)==' '))    //condicion de primera letra de una palabra
      if (*p>='a' && *p<='z')   //si es una letra minúscula
        *p = *p - ('a'-'A');    //también podría ser:  *p -= 'a'-'A';
    p++;   //p pasa a apuntar a la siguiente letra
  }
  printf("\nLa frase corregida es: %s", cad);
}
```

---

5. Hacer una función que reciba como argumento la dirección de comienzo de una cadena de caracteres, y de la que hay que eliminar las vocales (suponer solo letras minúsculas).
El prototipo de la función debe ser: **char *fun(char *c);**
La función tiene que devolver la misma dirección de comienzo de la cadena original, que ahora está sin vocales.

Solución:

```
char *fun(char *c)
{
  int i, j;
  for (i=0; c[i]!='\0'; i++)
  {  if (c[i]=='a' || c[i]=='e' || c[i]=='i' || c[i]=='o' || c[i]=='u')
     {  for (j=i; c[j]!='\0'; j++)
          c[j]=c[j+1];
        i--;
     }
  }
  return c;
}
```

Otra solución:

```
char *fun(char *c)
{
  char *ini=c, *aux;
```

```
  while (*c!='\0')
  {  if (*c=='a' || *c=='e' || *c=='i' || *c=='o' || *c=='u')
     {  aux=c;
        do
        {  aux++; *(aux-1)=*aux;  }
        while (*aux!='\0');
     }
     else c++;
  }
  return ini;
}
```

Otra solución:

```
char *fun(char *c)
{
  char *cad=(char *)malloc(strlen(c)+1);
  char *aux=cad, *ini=c;
  while (*c!='\0')
     if (*c=='a' || *c=='e' || *c=='i' || *c=='o' || *c=='u')  c++;
     else *aux++=*c++;
  *aux='\0';
  strcpy(ini,cad); free(cad);
  return ini;
}
```

---

6. Escribir las instrucciones que hay que sustituir por la línea de **/*COMENTARIO */** para que la función **maymin()** calcule el mayor y el menor de los elementos de una lista de números float.
El argumento 1º es la dirección de comienzo de la lista, el 2º es la dirección de memoria de la variable donde se ha de meter el mayor (pasada por referencia), el 3º la del menor y el 4º el número de elementos de la lista.

```
 void maymin(float *p, float *mayor, float *menor, int n)
 {
   *mayor = *menor = *p++;
   while(--n>0)
     /* COMENTARIO */
  }
```

Solución:

```
{
  if (*mayor<*p) *mayor=*p;
  if (*menor>*p) *menor=*p;
  p++;
}
```

---

7. Diseñar la función de nombre **quitarEspaciosYComas**, que copie en **textoSin** la cadena que hay en **textoCon** sin espacios ni comas.

```
int quitarEspaciosYComas(const char *textoCon, char *textoSin);
/* funcion que permite quitar espacios y comas de una cadena de longitud indefinida
 * [IN ] textoCon, array de char para eliminar espacios y comas
 * [OUT] textoSin, array de char para copiar la cadena sin espacios y comas
 * [RET] return, Si error devolver numero < 0 */
```

Solución:

```
int quitarEspaciosYComas(const char *textoCon, char *textoSin)
{
   unsigned int i;  //indice que recorre textoCon
   unsigned int j=0;  //indice que recorre textoSin
   int ret=0;

   if (textoCon != NULL && textoSin != NULL)
   {
    for(i=0; i<strlen(textoCon); i++)
    {
        if (textoCon[i] != ' ' && textoCon[i] != ',')
        {
           textoSin[j] = textoCon[i];  j++;
        }
    }

    textoSin[j] = '\0';  //añado el caracter de fin de cadena
    ret = 1;
   }
   return ret;
}
```

---

8. Diseñar una función de prototipo **void fun(char *inicio, char *final)** que reciba como argumentos de entrada dos cadenas de caracteres, la primera representando una hora y minuto de inicio (HH:MM), y la segunda una hora y minuto de final (HH:MM). La función debe calcular e imprimir por pantalla el nº total de minutos que hay entre una hora y la otra, y también el nº de Horas+Minutos. Por ejemplo, la función **fun("12:45", "15:15")** imprimirá en pantalla:

Minutos totales: 150 minutos Horas y Minutos: 2 horas 30 minutos

Si el formato de las cadenas de caracteres no es el correcto, se emitirá un mensaje de error: deben medir 5 caracteres, el del centro debe ser "**:**", los demás deben ser cifras del 0 al 9, la hora debe estar entre 00 y 23, y el minuto debe estar entre 00 y 59. Para detectar esto se creará y utilizará una función de prototipo **int estamal(char *cad)** que devuelve un **1** si la cadena está mal, o un **0** si está bien.

Si la segunda cadena representa una hora inferior a la primera, se emitirá también un mensaje de error.

Se puede utilizar la función **atoi()**, así como el operador **módulo (%)** que devuelve el resto de una división entera.

Aunque no es la única forma de hacerlo, para convertir la cadena de tipo HH:MM a horas y minutos se puede utilizar la función **sscanf** que lee variables desde una cadena de texto, de forma análoga a **fscanf** (que las lee desde un fichero) y **scanf** (que las lee del teclado).
Su prototipo es: **int sscanf(char *cadena, const char *formato [, argumentos]...);**
El argumento **cadena** es desde donde se leen los datos de entrada.

Solución:

```
int estamal(char *cad)
{
   int hora, minuto, i;
   if (strlen(cad) != 5) return 1;
   if (cad[2] != ':') return 1;
   for (i=0; i<5; i++)
      if (i!=2 && (cad[i]<'0' || cad[i]>'9')) return 1;
   hora = atoi(cad); minuto = atoi(cad+3);
   if (hora>23 || minuto>59) return 1;
   return 0;
}
```

```
void fun(char *inicio, char *final)
{
   int minutosini, minutosfin, minutostot;
   if (estamal(inicio) || estamal(final))
     {  printf("ERROR de formato de hora"); exit(0);  }
   minutosini = atoi(inicio)*60 + atoi(inicio+3);
   minutosfin = atoi(final)*60 + atoi(final+3);
   minutostot = minutosfin - minutosini;
   if (minutostot<0)
     {  printf(""ERROR: hora final menor que inicial"); exit(0);  }
   printf("Minutos totales: %d", minutostot);
   printf("\tHoras y Minutos: %d horas %d minutos\n", minutostot/60, minutostot%60);
}

// CON sscanf:
int estamal(char *cad)
{
   int hora, minuto, i, ret = 0;
   i = sscanf(cad, "%d:%d", &hora, &minuto);
   if (i==2)      // Formato correcto
   {
       if (hora>23 || minuto>59)  ret = 1;
   }
   return ret;
}

void fun(char *inicio, char *final)
{
   int h0, h1, m1, m0, minutostot;
   sscanf(inicio, "%d:%d", &h0, &m0);
   sscanf(final, "%d:%d", &h1, &m1);
   minutostot = h1*60+m1 - (h0*60+m0);
   if (minutostot>=0)
   {
       printf("Minutos totales: %d", minutostot);
       printf("\tHoras y Minutos: %d horas %d minutos\n", minutostot/60, minutostot%60);
   }
   else
   {    printf(""ERROR: hora final menor que inicial");    }
}
```

---

9. Diseñar la función "**dateCompare**" con prototipo **int dateCompare(char *date1, char *date2)** que reciba como argumentos de entrada dos fechas en formato de cadena de caracteres "**dia/mes/año**", las compare y devuelva el valor **1** si la primera fecha es más alta (posterior) a la segunda fecha, o bien devuelva **-1** si la primera fecha es más baja (anterior) que la segunda fecha, o bien devuelva el valor **0** si ambas fechas son idénticas.
Si el formato de alguna de las fechas no es correcto, la función devolverá **-2** e imprimirá en pantalla un mensaje de error. El formato correcto debe ser "**dd/mm/aaaa**" (el separador debe ser '**/**'), el año debe estar comprendido entre 1000 y 3000, el mes entre 1 y 12, y el día debe entre 1 y 31 (no importa el mes).
Se puede hacer uso de la funciones **atoi()** o **sscanf()**.

Ejemplos:
- dateCompare("07/10/2017", "01/12/2016") devuelve 1
- dateCompare("07/10/2017", "01/12/2018") devuelve -1
- dateCompare("07/10/2017", "07/10/2017") devuelve 0
- dateCompare("07/10/2017", "HOLA") imprime "ERROR" y devuelve -2

Solución:

```
int dateCompare(char *date1, char *date2)
{
```

```
  int dia1=atoi(date1), mes1=atoi(date1+3), anyo1=atoi(date1+6);
  int dia2=atoi(date2), mes2=atoi(date2+3), anyo2=atoi(date2+6);
  int resul=0;

  if (date1[2]!='/' || date1[5]!='/' || date2[2]!='/' || date2[5]!='/')  resul=-2;
  else if (dia1<1 || dia1>31 || mes1<1 || mes1>12 || anyo1<1000 || anyo1>3000)  resul=-2;
  else if (dia2<1 || dia2>31 || mes2<1 || mes2>12 || anyo2<1000 || anyo2>3000)  resul=-2;
  if (resul==-2)  {  printf("ERROR"); return resul;  }

  if (anyo1>anyo2)  resul=1;
  else if (anyo1<anyo2)  resul=-1;
    // Los años son iguales:
  else if (mes1>mes2)  resul=1;
  else if (mes1<mes2)  resul=-1;
    // Los meses también son iguales:
  else if (dia1>dia2)  resul=1;
  else if (dia1<dia2)  resul=-1;
  else  resul=0;
  return resul;
}
```

O BIEN CON sscanf():

```
int dateCompare(char *date1, char *date2)
{
  int dia1, mes1, anyo1, dia2, mes2, anyo2, result, r1, r2;

  r1 = sscanf(date1, "%d/%d/%d", &dia1, &mes1, &anyo1);
  r2 = sscanf(date2, "%d/%d/%d", &dia2, &mes2, &anyo2);

  if (r1!=3 || r2!=3)   // Comprobamos que los 3 argumentos tienen el formato y se han leído bien
     result = -2;
  else if (dia1<1 || dia2<1 || dia1 > 31 || dia2 > 31 || mes1 < 1 || mes2 < 1 || mes1 >12 || mes2 >12 || anyo1
< 1000 || anyo2 < 1000 || anyo1 > 3000 || anyo2 > 3000)   // Comprobamos rangos de dia, mes y año
     result = -2;
  else if (anyo1 > anyo2)         // Si el año es mayor => 1
     result = 1;
  else if (anyo1==anyo2 && mes1>mes2)    // Mismo año, mes mayor => 1
     result = 1;
  else if (anyo1==anyo2 && mes1==mes2 && dia1>dia2)   // Mismo año y mes, dia mayor => 1
     result = 1;
  else if (anyo1==anyo2 && mes1==mes2 && dia1==dia2)   // Misma fecha => 0
     result = 0;
  else
     result = -1;

  if (result == -2)       // Imprimir Error
     printf("\nERROR\n");
  return result;
}
```

ARRAYS DE PUNTEROS

1. Crear un programa que reciba desde el teclado una fecha introducida en format numérico (**25-12-2010**) y la muestre en pantalla en formato texto (**25 de diciembre de 2010**).
Se controla si los datos introducidos son correctos: número de día entre **1 y 28-29-30-31** (dependiendo del mes del año), número de mes entre **1 y 12**, año entre **1000 y 3000**. Si alguno de ellos es incorrecto, la fecha total es solicitada de nuevo.

Solución:

```
#include <stdio.h>
int diasMes(int mm, int aa);
char *nombreMes(int mm);

main( )
{ struct tipofecha { int dia, mes, anyo; } fecha;
  int mal;       //indica un error en los datos de entrada

  do
  { printf("Introduce la fecha (dd-mm-aaaa): ");
    scanf("%2d-%2d-%4d", &fecha.dia, &fecha.mes, &fecha.anyo);  //lee la fecha desde teclado
    mal = (fecha.dia<1 || fecha.dia>diasMes(fecha.mes, fecha.anyo));  //revisa el valor del dia
    mal = mal || (fecha.mes<1 || fecha.mes >12);              //revisa el valor del mes
    mal = mal || (fecha.anyo<1000 || fecha.anyo>3000);        //revisa el valor del año
  } while (mal);    //si algo está mal, pide la fecha de nuevo

  printf("La fecha es: %d de %s de %d", fecha.dia, nombreMes(fecha.mes), fecha.anyo);
}

int diasMes(int mm, int aa)     //retorna el nº de dias del mes mm
{
  int d=0;
  switch (mm)
  { case 1: case 3: case 5: case 7: case 8: case 10: case 12:  //meses con 31 dias
      d = 31; break;
    case 4: case 6: case 9: case 11:  // meses con 30 dias
      d = 30; break;
    case 2:                       //Febrero
      if (aa%4==0 && aa%100!=0 || aa%400==0)  //formula para saber si un año es bisiesto
        d = 29;
      else d = 28;
  }
  return d;
}

char *nombreMes(int mm)    //retorna el nombre del mes mm
{
  char *mes[ ] = {"Mes incorrecto","enero","febrero","marzo","abril","mayo","junio","julio",
"agosto","septiembre","octubre","noviembre","diciembre"};
  return (mm>0 && mm<13) ? mes[mm] : mes[0];     //retorna el punter correcto
}
```

PASO DE ARGUMENTOS POR REFERENCIA

1. Crear una función con prototipo **void maymin(float *p, float *mayor, float *menor, int n);** que calcule el valor mayor y el menor de los elementos de un array de números de tipo **float**.
El argumento **p** es la dirección de comienzo del array, el argumeto **mayor** es la dirección de memoria donde se ha de depositar el valor mayor, el argumento **menor** la del valor menor y el argumento **n** es el número de elementos del array.

Solución:

```
void maymin(float *p, float *mayor, float *menor, int n)
```

```
{
  *mayor=*menor=*p++;
  while(--n>0)
  {
    if (*mayor<*p) *mayor=*p;
    if (*menor>*p) *menor=*p;
    p++;
  }
}
```

---

2. Crear una función **"func"** que simule la devolución de dos resultados: recibe desde el programa **main()** dos argumentos de entrada numéricos **a** y **b** leídos desde el teclado y retorna dos resultados almacenados en dos variables **resul1** y **resul2**, variables locales en **main()**. Los resultados son: **resul1=a+b, resul2=a/b** (con decimales). Para ello, ambas variables son pasadas *"por referencia"* a la función **func**, además de los dos datos **a** y **b**, y la función **func** deposita en ellas los dos resultados solicitados.

Solución:

```
#include <stdio.h>
void func(int a, int b, int *pr1, float *pr2);
main( )
{ int a=0, b=0;                  //datos de entrada
  int resul1; float resul2;   //variables de resultado
  printf("Introduce los 2 datos de entrada:");
  scanf("%d %d", &a, &b);
  func(a, b, &resul1, &resul2);   //pasamos por referencia las dos variables de resultado
  printf("\nLos 2 resultados son: %d, %g", resul1, resul2);
}

void func(int a, int b, int *pr1, float *pr2)
{ *pr1 = a+b;
  *pr2 = (b==0)?0:(float)a/b;   //si b==0, retornamos 0
}
```

ASIGNACIÓN DINÁMICA DE MEMORIA

1. Indique por qué instrucciones se deben sustituir las 4 líneas de comentario **/* prototipo */** , **/* llamada a fun() */** , **/* función fun() */** y **/* liberar memoria */** para que el programa funcione correctamente. El programa debe servir para crear un array dinámico de **n** datos de tipo **int** e introducir por teclado sus valores.

```
/* prototipo */
main()
{ int *x, n=0, i=0;
  printf("Indique el número de elementos del array: ");
  scanf("%d",&n);
  /* llamada a fun() */
  while(i++<n) scanf("%f", x[i]);
  /* liberar memoria */
}
/* funcion fun() */
```

Solución:

```
/* prototitpo */:
int *fun(int);
```

```
/* llamada a fun() */:
x=fun(n);

/* liberar memoria */:
free(x);

*/ funcion fun() */:
int *fun(int n)
{  int *p=(int *)malloc(n*sizeof(int));
   return p;
}
```

---

2. Crear una función con prototipo **char *fun(char *s);** que permita hacer un duplicado de una cadena de caracteres dada por **s**. La función ha de devolver la dirección de la nueva cadena creada.

Solución:

```
char *fun(char *s)
{
   char *p=(char *)malloc(strlen(s)+1);
   strcpy(p,s);
   return p;
}
```

---

3. Escribir un programa que, usando punteros, haga algo parecido a la function **"strcat"** (concatenación de cadenas). El programa acepta por teclado dos cadenas de caracteres (**cad1** y **cad2**, máximo 100 caracteres cada una) y crea mediante asignación dinámica de memoria una tercera cadena **cad3** con la minima longitud necesaria para acomodar ambas cadenas concatenadas **cad1+cad2**, luego copia a **cad3** todos los caracteres de **cad1** seguidos de los de **cad2**, y finalmente muestra el resultado en la pantalla (ambas cadenas iniciales concatenadas). Finálmente libera la memoria reservada.
Para este programa, crear una función auxiliar llamada **"StrLen"** que hace lo mismo que la funcion estandar **"strlen"**, es decir, retorna el número de caracteres incluidos en una cadena de caracteres.

Solución:

```
int Strlen(char *cad);

main( )
{ char cad1[100], cad2[100], *cad3;
  int i, j=0, n;

  printf("Teclea la primera cadena: "); gets(cad1);
  printf("Teclea la segunda cadena: "); gets(cad2);

  n = Strlen(cad1)+Strlen(cad2)+1;  //n = longitud de la cadena concatenada

  cad3 = (char *)malloc(n);   //reserve dinamica de memoria para cad3
  if (cad3==NULL) { printf("Error: memoria llena"); exit(0); }

  for (i=0; i<Strlen(cad1); i++)  //copia cad1 en cad3
    { cad3[j] = cad1[i]; j++; }
  for (i=0; i<Strlen(cad2); i++)  //copia cad2 a continuación en cad3
    { cad3[j] = cad2[i]; j++; }
  cad3[j] = 0;   //caracter \0 al final de la cadena concatenada

  printf("\nLas cadenas concatenadas son:\n%s", cad3);
```

```
  free(cad3);   //liberar memoria antes de terminar el programa
}

int Strlen(char *cad)  //retorna el número de caracteres dentro de una cadena
{ int n;
  for (n=0; cad[n]!=0; n++);  //n se incrementa hasta que se alcanza el character \0 final
  return n;
}
```

---

4. Realizar un programa que cree en memoria un array dinámico unidimensional de datos de tipo **double**. El programa pedirá inicialmente por teclado el número deseado de elementos de tal array, luego lo creará mediante asignación dinámica de memoria, a continuación pedirá por teclado los datos a almacenar en dichos elementos del array, luego calculará la media aritmética de dichos datos, la presentará en pantalla y finálmente eliminará el array, liberando la memoria utilizada por el mismo.

Solución:

```
main()
{ double *p, media=0;
  int numelem=0, i;
  do
  { printf("Dime Nº de elementos del array a crear:"); scanf("%d", &numelem);
  } while (numelem<1);

  p = (double *)malloc(numelem*sizeof(double));   //Crear array dinámico
  if (p==NULL)  { printf("Error: No hay memoria suficiente."); exit(0); }

  printf("Introduzca los datos del array:\n");
  for (i=0; i<numelem; i++)  scanf("%lf", &p[i]);   //Leer datos del array

  for (i=0; i<numelem; i++)  media+=p[i];     //Calcular la media
  media = media/numelem;
  printf("\nLa media aritmética es: %lg\n", media);

  free(p);    //Liberar memoria
}
```

---

5. Diseñar una función **fun()** que reciba como argumentos de entrada dos cadenas de caracteres **cad1** y **cad2**, y que genere otra cadena que contenga la concatenación de aquellas dos (**cad1** seguida de **cad2**). La función devolverá como resultado la dirección de memoria donde reside dicha cadena recien creada. Se puede hacer uso de las funciones estandar de tratamiento de cadenas (**strcpy(), strcat(), strlen()...)**.
El prototipo de la función será: **char *fun(char *cad1, char *cad2);**

Solución:

```
char *fun(char *cad1, char *cad2)
{
  char *aux;  int tam;
  tam = strlen(cad1)+strlen(cad2)+1;     // +1 para el caracter \0  final
  aux = (char *)malloc(tam);              //Crear la nueva cadena
  if (aux!=NULL)
  { strcpy(aux,cad1); strcat(aux,cad2); }    //Copiar cad1+cad2 en aux
  return aux;
}
```

---

6. Escribir una función con prototipo **float **fun(int ff, int cc);** que cree en la memoria un array dinámico bidimensional de números reales float de **ff** filas y **cc** columnas mediante asignación dinámica de memoria:

1º. Creará en la memoria el array, gobernado por el puntero **float **array;** de forma que sus elementos puedan ser accedidos mediante dos subíndices (**array[i][j]**).
2º. Cargará todo el array con datos numéricos reales introducidos por teclado, rellenándolo por filas.
3º. Presentará en pantalla todos estos datos introducidos, en forma de tabla (imprimiendo por filas).
4º. Devolverá el puntero **array** que gobierna al array bidimensional.

Solución:

```
float **fun(int ff, int cc)
{
  float **array;
  int i,j;
  // Creación del array
  array = (float **)malloc(ff * sizeof(float *));
  if (array==NULL)  { printf("ERROR: falta memoria."); exit(0); }
  for (i=0; i<ff; i++)
  {   array[i] = (float *)malloc(cc * sizeof(float));
      if (array[i]==NULL)  { printf("ERROR: falta memoria."); ff=i; }
  }
  // Toma de datos desde el teclado
  printf("\nIntroduce los datos por teclado:\n");
  for (i=0; i<ff; i++)
     for (j=0; j<cc; j++)
     {   printf("array[%d][%d]=", i, j); scanf("%f", &array[i][j]);   }
  // Presentación de datos en pantalla
  printf("\nLos datos introducidos son:\n");
  for (i=0; i<ff; i++)
  {  for (j=0; j<cc; j++)   printf("%g\t", array[i][j]);
     printf(("\n");
  }
  // Valor retornado
  return array;
}
```

---

7. Con la siguiente declaración:
typedef struct ficha { char nombre[40]; int edad; float euros; } tficha;

Escribir una función cuyo prototipo sea **tficha *fun(int n);** que reserve en memoria un array dinámico de **n** estructuras utilizando la función **malloc()**, y que devuelva como resultado la dirección de comienzo de dicho array, o bien NULL si no hay memoria libre suficiente.

Solución:

```
tficha *fun(int n)
{
  tficha *pun;
  pun = (tficha *)malloc(sizeof(tficha)*n);
  if (pun==NULL) printf("No hay memoria.");
  return pun;
}
```

---

8. Escribir un programa que cree en la memoria, mediante asignación dinámica de memoria, un array dinámico de estructuras del siguente tipo:
typedef struct ficha { char nombre[40]; int edad; float euros; } tficha;

1º. Solicitará por teclado el tamaño deseado del array (nº de estructuras a almacenar).

2º. Creará en la memoria el array con **calloc**, gobernado por el puntero **tficha *array;**
3º. Cargará todo el array con datos introducidos por teclado, mediante un bucle.
4º. Presentará en pantalla todos estos datos introducidos.
5º. Eliminará de la memoria todo el array, antes de terminar el programa.

Solución:

```
typedef struct ficha  {  char nombre[50];  int edad;  float euros;  } tficha;
int main()
{
  tficha *array;
  int i, num=0;
  // Tomar nº de elementos desde el teclado
  do
  {   printf("Dime nº de estructuras: "); scanf("%d", &num);  }
  while (num<=0);
  // Creación del array
  array = (tficha *)calloc(num, sizeof(tficha));
  if (array==NULL)  { printf("ERROR: falta memoria."); exit(0); }
  // Toma de datos desde el teclado
  printf("\nIntroduce los datos por teclado:\n");
  for (i=0; i<num; i++)
  {   printf("\nEstructura Nº %d:\n", i); fflush(stdin);
      printf("Nombre: "); gets(array[i].nom);
      printf("Edad: "); scanf("%d", &array[i].edad);
      printf("Euros: "); scanf("%f", &array[i].euros);
  }
  // Presentación de datos en pantalla
  printf("\nLos datos introducidos son:\n");
  for (i=0; i<num; i++)
      printf("Nombre: %s\tEdad: %d\tEuros: %g\n", array[i].nombre, array[i].edad, array[i].euros);
  // Borrado del array
  free(array);
}
```

---

9. Escribir un programa que cree en la memoria un array dinámico bidimensional de números reales **float** mediante asignación dinámica de memoria:
1º. Solicitará por teclado el tamaño deseado del array (nº de filas y nº de columnas).
2º. Creará en la memoria el array, gobernado por el puntero **float **array;** de forma que sus elementos puedan ser accedidos mediante dos subíndices (**array[i][j]**).
3º. Cargará todo el array con datos numéricos reales introducidos por teclado, rellenándolo por filas.
4º. Presentará en pantalla todos estos datos introducidos, en forma de tabla (por filas).
5º. Eliminará de la memoria todo el array, antes de terminar el programa.

Solución:

```
int main()
{
  float **array;
  int fil=0,col=0,i,j;
  do
  {   printf("Dime nº de filas: "); scanf("%d", &fil);  }
  while (fil<=0);
  do
  {   printf("Dime nº de columnas: "); scanf("%d", &col);  }
  while (col<=0);
  // Creación del array
  array = (float **)malloc(fil * sizeof(float *));
  if (array==NULL)  { printf("ERROR: falta memoria."); exit(0); }
```

```
  for (i=0; i<fil; i++)
  {   array[i] = (float *)malloc(col * sizeof(float));
      if (array[i]==NULL)  { printf("ERROR: falta memoria."); fil=i; }
  }
  // Toma de datos desde el teclado
  printf("\nIntroduce los datos por teclado:\n");
  for (i=0; i<fil; i++)
     for (j=0; j<col; j++)
     {   printf("array[%d][%d]=", i, j); scanf("%f", &array[i][j]);   }
  // Presentación de datos en pantalla
  printf("\nLos datos introducidos son:\n");
  for (i=0; i<fil; i++)
  {  for (j=0; j<col; j++)   printf("%g\t", array[i][j]);
     printf(("\n");
  }
  // Borrado del array
  for (i=0; i<fil; i++)  free(array[i]);
  free(array);
}
```

---

10. Dadas las declaraciones de **struct tiempo** y **struct ciclista** aquí indicadas, complete el programa para que controle una carrera ciclista y realice lo siguiente:
1º. Pregunte cuántos ciclistas hay en la carrera y lo guarde en la variable **num**.
2º. Cree un array de nombre **c** con **num** estructuras de tipo **struct ciclista** mediante asignación dinámica de memoria.
3º. Mediante un bucle, lea desde teclado los registros de dicho array (nombres de los ciclistas y sus tiempos). Esto debe implementarse en la función **leeDatos**.
4º. Ordene el array de menor a mayor tiempo de carrera. Para esto, diseñar y utilizar una función que devuelva el nº total de segundos a partir de un tiempo horas-minutos-segundos, con el siguiente prototipo:
int segundos(struct tiempo t);
5º. Presente en pantalla el array ordenado, y libere la memoria dinámica utilizada.

```
struct tiempo { int horas, minutos, segundos; };
struct ciclista { char nombre[40]; struct tiempo tmp; };

int segundos(struct tiempo t);
void leeDatos (struct ciclista *c, int num);
void ordenaDatos (struct ciclista *c, int num);
void imprimirDatos(struct ciclista *c, int num);

main()
{
  struct ciclista *c;  int num;

  //1º: preguntar cuántos ciclistas y guardarlo en num

  //2º: reservar memoria para c

  //3º: llamar a la función leeDatos

  //4º: llamar a la función ordenaDatos

  //5º: llamar a la función imprimirDatos y liberar memoria

}

int segundos(struct tiempo t)
{  //Complete esta función:
}
```

```
void leeDatos (struct ciclista *c, int num)
{ //Complete esta función:
}

void ordenaDatos (struct ciclista *c, int num)
{ //Complete esta función:
}

void imprimirDatos(struct ciclista *c, int num)
{ //Complete esta función:
}
```

Solución:

```
  //1º: preguntar cuántos ciclistas y guardarlo en num
  printf("Dime Nº de ciclistas: ");
  scanf("%d", &num);

  //2º: reservar memoria para c
  c = (struct ciclista *)malloc(num*sizeof(struct ciclista));
  if (c==NULL) { printf("\nFalta memoria\n"); exit(0); }

  //3º: llamar a la función leeDatos
  leeDatos(c, num);

  //4º: llamar a la función ordenaDatos
  ordenaDatos(c, num);

  //5º: llamar a la función imprimirDatos y liberar memoria
  imprimirDatos(c, num);
  free(c);
}

int segundos(struct tiempo t)
{  //Complete esta función:
     return (t.horas*3600 + t.minutos*60 + t.segundos);
}

void leeDatos (struct ciclista *c, int num)
{ //Complete esta función:
  int i;
  for (i=0; i<num; i++)
  {
    printf("\nCiclista nº %d:\n", i+1);
    printf("Nombre: "); gets(c[i].nombre);
    printf("Tiempo (hh:mm:ss): ");
    scanf("%d:%d:%d", &c[i].tmp.horas, &c[i].tmp.minutos, &c[i].tmp.segundos);
  }
}

void ordenaDatos (struct ciclista *c, int num)
{ //Complete esta función:
 struct ciclista aux;  int i, j;
 for (i=0; i<num-1; i++)
    for (j=i+1; j<num; j++)
       if (segundos(c[i].tmp) > segundos(c[j].tmp))
       {  aux=c[i]; c[i]=c[j]; c[j]=aux;  }
}
```

```
void imprimirDatos(struct ciclista *c, int num)
{ //Complete esta función:
  int i;
  printf("\nLista ordenada:\n");
  for (i=0; i<num; i++)
    printf("Nombre: %s\tTiempo: %d:%d:%d\n", c[i].nombre, c[i].tmp.horas,
    c[i].tmp.minutos, c[i].tmp.segundos);
}
```

PASO DE ARRAYS A FUNCIONES

1. Realizar una función cuyo prototipo es **void ordenar(int *, int);** que ordene un array de enteros. La función **ordenar()** recibe como primer argumento el nombre del array y como segundo argumento el número de elementos del array.

Solución:

```
void ordenar(int *p, int n)
{ int i,j,aux;
  for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
      if (p[i]>p[j])  { aux=p[i]; p[i]=p[j]; p[j]=aux;  }
}
```

---

2. Crear un programa que declara un array bidimensional local en la función **main()** y lo pasa a 3 funciones sucesivas:

1. Función **inputdata** que llena el array con datos desde el teclado.
2. Función **sumrows** que obtiene y presenta en pantalla las sumas de cada fila del array.
3. Función **sumcolumns** que obtiene y presenta en pantalla las sumas de cada columna del array.

Solución:

```
#include <stdio.h>
void inputdata(float a[ ][3]);
void sumrows(float a[ ][3]);
void sumcolumns(float a[ ][3]);

main( )
{ float array[3][3];                  //array principal
  printf("Introduce los datos del array:\n");   inputdata(array);
  printf("Las sumas de cada fila son las siguientes:\n");   sumrows(array);
  printf("Las sumas de cada columna son las siguientes:\n");   sumcolumns(array);
}

void inputdata(float a[ ][3])
{ int i, j;
  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
        {  printf("array[%d][%d] = ", i, j);  scanf("%f", &a[i][j]);  }
}

void sumrows(float a[ ][3])
{ int i, j;  float sum;
  for (i=0; i<3; i++)
  {  for (j=0, sum=0; j<3; j++)
          sum += a[i][j];
    printf("Sum of row %d = %g\n", i, sum);
  }
```

```
}

void sumcolumns(float a[ ][3])
{ int i, j;  float sum;
  for (j=0; j<3; j++)
  {  for (i=0, sum=0; i<3; i++)
          sum += a[i][j];
     printf("Sum of column %d = %g\n", j, sum);
  }
}
```

PASO DE ESTRUCTURAS A FUNCIONES

1. Con las siguientes declaraciones:

```
struct ficha
{ char nom[40];
  float nota;
} x[10];
int n;
```

La instrucción: **printf("\n %s", mayor(x,n));** debe imprimir el nombre del alumno de mayor nota del array **x** (suponer que no hay notas repetidas), array que tiene un número de elementos dado por **n**. Diseñar la función **mayor().**

Solución:

```
char *mayor(struct ficha *x, int n)
{
  int i=0,m=0;
  for (i=1; i<n; i++)
    if (x[i].nota>x[m].nota)
      m=i;
  return x[m].nom;
}
```

---

2. Se ha declarado la estructura de datos **DatosPersonales_t** indicada, y en la función **main** se ha declarado un array de dichas estructuras llamado **arrayDPersonales** de 100 elemenos, así como una variable entera **maxElem** que indicará en todo momento el número de elementos del array que contienen datos válidos (no tiene por qué ser el máximo de 100).
Suponga que los datos ya han sido introducidos en el array desde la función **main**.
Implemente una función llamada **imprimeDatosPersonales** que imprima por pantalla todos los datos personales del array. Indique también cómo se realizará la llamada a dicha función desde la función **main**.

```
typedef struct
{  char nombre[100];   // nombre y apellidos persona
   char dni[9];          // DNI
   int edad;             // edad
   char sexo;            // 'H' para hombre y 'M' para mujer
} DatosPersonales_t;
...
void main()
{
   DatosPersonales_t  arrayDPersonales[100];
   int maxElem;
   ...
   // llamada a la funcion aquí:
   ...
```

```
}
// Implementación de la funcion aquí:
```

Solución:

```
  // llamada a la funcion aquí:
  imprimeDatosPersonales(arrayDPersonales, maxElem);

// Implementación de la funcion aquí:
void  imprimeDatosPersonales(DatosPersonales_t *datos, int elem)
{
   int i;
   for (i=0; i<elem; i++)
   {
       printf ("Nombre: %s, DNI: %s, EDAD: %d, SEXO: %c\n",
                datos[i].nombre, datos[i].dni, datos[i].edad, datos[i].sexo);
   }
}
```

---

3. Se ha declarado la estructura de datos **DatosPersonales_t** indicada, y en la función **main** se ha declarado un array de dichas estructuras llamado **arrayDPersonales** de 100 elementos, así como una variable entera **maxElem** que indicará en todo momento el número de elementos del array que contienen datos válidos (no tiene por qué ser el máximo de 100).
Suponga que los datos ya han sido introducidos en el array desde la función **main**.
Implemente una función llamada **imprimeDatosPersonales** que reciba el array **arrayDPersonales** como argumento de entrada y que imprima por pantalla todos los datos personales de dicho array. Indique también cómo se realizará la llamada a dicha función desde la función **main**.

```
typedef struct
{  char nombre[100];   // nombre y apellidos persona
   char dni[9];          // DNI
   int edad;             // edad
   char sexo;            // 'H' para hombre y 'M' para mujer
} DatosPersonales_t;
...
void main()
{
   DatosPersonales_t  arrayDPersonales[100];
   int maxElem;
   ...
   // Incluir la llamada a la funcion AQUÍ:
  ...
}

// Implementación de la funcion AQUÍ:
```

Solución:

```
// Incluir la llamada a la funcion AQUÍ:
imprimeDatosPersonales(arrayDPersonales, maxElem);

// Implementación de la funcion AQUÍ:
void  imprimeDatosPersonales(DatosPersonales_t *datos, int elem)
{
   int i;
   for (i=0; i<elem; i++)
   {
       printf ("Nombre: %s, DNI: %s, EDAD: %d, SEXO: %c\n",
```

```
            datos[i].nombre, datos[i].dni, datos[i].edad, datos[i].sexo);
    }
}
```

---

4. Dadas las siguientes declaraciones:

```
struct tiempo_t
{
        int horas;
        int minutos;
        int segundos;
};
typedef struct
{
        char nombre[50];
        int anyoNacimiento;
        struct tiempo_t tiempo;
} atleta_t;
```

Escribir un programa que cree, mediante asignación dinámica de memoria, un array de estructuras de tipo **atleta_t**. Escribir para ello las funciones **main**, **crearArray** y **mostrarArray** de acuerdo con las siguientes indicaciones:

1º. Solicitará por teclado el tamaño deseado del array (número de estructuras a almacenar).
2º. Reservará memoria para el array con **malloc** mediante una función cuyo prototipo debe ser:
        atleta_t *crearArray(int numAtletas);
3º. Además, esta función rellenará mediante un bucle todos los campos numéricos a cero y el campo nombre con una cadena vacía (**""**). La función devolverá como resultado la dirección de comienzo del array, o bien NULL si no se ha podido reservar la memoria.
4º. Presentará en pantalla todos los datos del array de estructuras mediante una función cuyo prototipo debe ser:
        void mostrarArray(atleta_t *arrayAtletas, int numAtletas);
5º. Finalmente, liberará de la memoria todo el array, antes de terminar el programa.

Solución:

```
atleta_t *crearArray(int numAtletas)
{
   int i=0;
   atleta_t *arrayAtletas = (atleta_t *) malloc (numAtletas * sizeof(atleta_t));

   if (arrayAtletas != NULL)
   {
      for(i=0; i<numAtletas; i++)
      {
         strcpy(arrayAtletas[i].nombre,"");
         arrayAtletas[i].anyoNacimiento=0;
         arrayAtletas[i].tiempo.horas=0;
         arrayAtletas[i].tiempo.minutos=0;
         arrayAtletas[i].tiempo.segundos=0;
      }
   }
   return arrayAtletas;
}

void mostrarArray(atleta_t *arrayAtletas, int numAtletas)
{
   int i=0;
   if (arrayAtletas!=NULL && numAtletas>0)
   {
```

```
      for(i=0; i<numAtletas; i++)
      {
         printf("Atleta %s\n\tAño de nacimiento: %d\n\tMejor tiempo: %d:%d:%d\n",
arrayAtletas[i].nombre, arrayAtletas[i].anyoNacimiento, arrayAtletas[i].tiempo.horas,
arrayAtletas[i].tiempo.minutos, arrayAtletas[i].tiempo.segundos);
      }
   }
}

void main()
{
   int numAtletas=0;
   atleta_t *arrayAtletas=NULL;
   printf("Introduce numero de atletas: ");
   scanf("%d",&numAtletas);

   arrayAtletas=crearArray(numAtletas);

   if (arrayAtletas!=NULL)
   {
      mostrarArray(arrayAtletas,numAtletas);
   }
   free(arrayAtletas);
}
```

ARGUMENTOS EN LÍNEA DE ÓRDENES

1. Utilizando únicamente las declaraciones que aparecen en el encabezamiento de la función main: **main(int argc, char **argv)** (sin ninguna variable adicional), realizar un programa que imprima cada uno de los argumentos de la línea de órdenes (sin incluir el nombre del programa ejecutable).

Solución:

```
#include <stdio.h>
main(int argc, char **argv)
{
  while (++argv)
    printf("%s\n", *argv);
}
```

---

2. Escribir un programa que admita argumentos en la linea de comandos, que imprima en pantalla todos los argumentos pasados en la linea de comandos menos el primero. Los imprimirá en orden ascendente (mismo orden de entrada) si el primer argumento pasado es **"a"**, o bien en orden descendente (orden contrario al de entrada) si el primer argumento pasado es **"d"**.
Por ejemplo, si el programa ejecutable se llama **prog** y damos la orden **prog a rojo verde azul**, imprimirá en pantalla **rojo verde azul**, y si damos la orden **prog d rojo verde azul** debe imprimir **azul verde rojo**.
El programa verificará que se han recibido al menos dos argumentos en la línea de comandos. En caso contrario emitirá un mensaje de error y el programa terminará.

Solución:

```
int main(int argc, char *argv[ ])
{  int i;
  if (argc<3)
    { puts("Programa mal usado"); exit(1); }
```

```
  if (argv[1][0]=='a')          // o bien:  if (strcmp(argv[1],"a")==0)
                                // o bien:  if (*argv[1]=='a')
    for (i=2; i<argc; i++)
      printf("%s ", argv[i]);

  else if (argv[1][0]=='d')     // o bien:  if (strcmp(argv[1],"d")==0)
                                // o bien:  if (*argv[1]=='d')
    for (i=argc-1; i>1; i--)
      printf("%s ", argv[i]);

  else
    printf("Mal, elegir a o d como primer argumento.\n");
}
```

---

3. Hacer un programa que imprima en pantalla todos los argumentos pasados desde la linea de órdenes (sin incluir el nombre del programa ejecutable), imprimiendo cada argumento con la primera letra en mayúsculas y las demás en minúsculas. Suponer que se dispone de las funciones **strlwr()** y **toupper()**.

Solución:

```
main(int argc, char *argv[ ])
{  int i;
   printf("Los argumentos pasados son:\n");
   for (i=1; i<argc; i++)
   {   strlwr(argv[i]);
       argv[i][0] = toupper(argv[i][0]);
       printf("%s  ", argv[i]);
   }
}
```

---

4. Hacer un programa con argumentos en línea de órdenes que imprima por pantalla los argumentos que se le pasan pero en letras mayúsculas. Por ejemplo, si el programa generado se llama **a.out**, la orden **./a.out Examen de Ciencias** imprimirá **EXAMEN DE CIENCIAS**.
Suponer que se dispone de la función **strupr()**, que convierte una cadena a mayúsculas.

Solución:

```
main(int argc, char **argv)
{
   while(*++argv != NULL)
      printf("%s ", strupr(*argv));
}
```

Otra solución:

```
main(int argc, char *argv[ ])
{  int i;
   for (i=1; i<argc; i++)
     printf("%s ", strupr(argv[i]));
}
```

---

5. Escribir un programa que acepte argumentos en la línea de comandos, y que acepte 2 argumentos: el primero será una **cadena** de caracteres sin espacios en blanco, y el segundo será un único **carácter**. El programa debe contar cuántas veces aparece este **carácter** (argumento 2º) dentro de la **cadena** (argumento 1º), y mostrarlo en pantalla. El programa emitirá un mensaje de error si se recibe un número de argumentos distinto de 2, o bien si el 2º argumento tiene más de un carácter de longitud.
Ejemplos: si el programa una vez compilado se llama "**prog**":

Si ejecutamos **./prog Salamanca a** el programa imprimirá: **El caracter a aparece 4 veces**
Si ejecutamos **./prog Salamanca u** el programa imprimirá: **El caracter u aparece 0 veces**
Si ejecutamos **./prog Salamanca** el programa imprimirá: **ERROR: deben ser 2 argumentos**
Si ejecutamos **./prog Salamanca ma** el programa imprimirá: **ERROR: el 2º argumento debe ser un unico caracter**

Solución:

```
int main(int argc, char *argv[ ])
{
   int contador=0, i;
   if (argc!=3)
      printf("ERROR: deben ser 2 argumentos\n");
   else if (strlen(argv[2])!=1)
      printf("ERROR: el 2º argumento debe ser un unico caracter\n");
   else
   {   for (i=0; i<strlen(argv[1]); i++)
         if (argv[1][i] == argv[2][0])  contador++;
       printf("El caracter %c aparece %d veces\n", argv[2][0], contador);
   }
}
```

O BIEN:

```
int main(int argc, char **argv)
{
   int contador=0, i;
   if (argc!=3)
   {  printf("ERROR: deben ser 2 argumentos\n"); exit(0);  }
   if (strlen(argv[2])!=1)
   {  printf("ERROR: el 2º argumento debe ser un unico caracter\n"); exit(0);  }
   for (i=0; argv[1][i] != '\0'; i++)
      if (argv[1][i] == *argv[2])  contador++;
   printf("El caracter %c aparece %d veces\n", *argv[2], contador);
}
```

---

6. Escribir un programa que calcule la letra de un número de DNI pasado como argumento por la línea de comandos. Además mostrará por pantalla el DNI completo.

Ejemplos: si el programa una vez compilado se llama "**letraDNI**":

- Si ejecutamos **./letraDNI** el programa imprimirá: **ERROR: Uso ./letraDNI numDNI**
- Si ejecutamos **./letraDNI 3085148** el programa imprimirá:

La letra del DNI 3085148 es C
El DNI completo es 03085148C

La operación para calcular la letra del DNI es como sigue:

- Dividir el número de DNI entre 23 y tomar el RESTO de la operación.
- Con el resto se puede saber la letra según la siguiente tabla.

Resto	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Letra	T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

En el ejemplo 3085148 / 23 = 134136 y de resto da 20, luego la letra es C.
Para solucionar el problema puedes utilizar el siguiente array donde, accediendo con el resto como índice, puedes obtener la letra:

char letra[ ]="TRWAGMYFPDXBNJZSQVHLCKE";

Para ello hay que programar las siguientes funciones:

void mostrarLetraDNI(int n)
Muestra por pantalla el texto: “La letra del DNI xxxxxxx es X”.

char *crearCadenaDNI(int n)
Reserva memoria con **malloc** para la cadena del DNI completo, la rellena con el número y la letra, y la devuelve para que sea mostrada en pantalla desde **main()**. Si el número de DNI tiene menos de 8 dígitos, se rellenará con ceros por la izquierda.

La función **main**
Recibe el argumento desde la linea de comandos y utiliza las dos funciones anteriores. Finalmente liberará la memoria reservada por la función **crearCadenaDNI**.

Para convertir un número a una cadena, se puede utilizar la función **sprintf** que escribe una cadena con formato depositándola en una cadena de texto, de forma análoga a como hace **fprintf** (que la deposita en un fichero) y **printf** (que la muestra por pantalla).
Su prototipo es: **int sprintf(char *cadena, const char *formato [, argumentos]...);**
El argumento **cadena** es donde se depositará la cadena de resultado.

Solución:

```
void main(int argc, char *argv[ ])
{
   int dni;
   if (argc!=2)
   {   printf("ERROR: Uso ./letraDNI numDNI \n"); exit(0);  }
   dni = atoi(argv[1]);
   char *p=NULL;

   mostrarLetraDNI(dni);
   p = crearCadenaDNI(dni);
   printf("\nEl DNI completo es %s\n", p);
   free(p);
}

void mostrarLetraDNI(int n)
{
   char letra[]="TRWAGMYFPDXBNJZSQVHLCKE";
   printf("\nLa letra del DNI %d es %c\n", n, letra[n%23]);
}

char *crearCadenaDNI(int n)
{
   char letra[]="TRWAGMYFPDXBNJZSQVHLCKE";
   char *p_c = malloc(10);
   sprintf(p_c, "%08d%c", n, letra[n%23]);
   p_c[10]='\0';
   return p_c;
}
```

---

7. Se quiere implementar un programa que reciba como parámetros de entrada desde la línea de comandos el nombre y la edad de una serie de personas, por ejemplo:

./programa Ana 24 Pep 78 Lucía 89 Daniel 45

Dicha información se debe almacenar en un array de estructuras como el siguiente:

```
struct Datos_t
{
  int edad;
  char nombre[50];
};
```

Desde la función **main()**, se debe comprobar que los datos introducidos son pares (nombre y edad de **N** individuos), se debe reservar memoria para el array de estructuras, y se debe llamar a la función **initDatos** para los datos de los N individuos. Una vez almacenados, se imprimirán por pantalla y se liberará la memoria. Implemente la función **main()** y la función **initDatos** según el siguiente prototipo:

```
void initDatos(struct Datos_t *misdatos, int posicion, char *nombre, int edad);

/* Función que guarda los datos pasados como argumento en un array de estructuras
* en la posición indicada por el argumento posicion
*
* misdatos  Puntero al array de datos
* posicion  Posicion dentro del array de datos donde se guardarán los datos
* nombre    Array de caracteres con el nombre a guardar
* edad      Entero con la edad a guardar
*/
```

Asumir que los argumentos pasados por la línea de comandos se introducen correctamente. Es decir, no hace falta comprobar que el nombre es una cadena de caracteres de longitud menor que 50 ni que la edad es un número.

Solución:

```
struct Datos_t
{
  int edad;
  char nombre[50];
};

void initDatos(struct Datos_t *misdatos, int posicion, char *nombre, int edad)
{
   misdatos[posicion].edad=edad;
   strcpy(misdatos[posicion].nombre, nombre);
}

int main (int argc, char *argv[ ])
{
   int numDatos = argc-1;
   int numIndividuos,  i;
   struct Datos_t *misDatos;   // Puntero al array de datos

   if (numDatos==0 || numDatos%2!=0)
   {
     printf("Los datos introducidos no son pares. Ejemplo: \n");
     printf (" ./prog Ana 24 Ramon 56 Pepe 101\n\n");
     exit (-1);
   }

   numIndividuos = numDatos/2;
   misDatos = (struct Datos_t *)malloc(numIndividuos*sizeof(struct Datos_t));   // Reservar memoria
```

```
   for (i=0; i<numIndividuos; i++)    // Carga de los N individuos en el array
     initDatos (misDatos, i,  argv[i*2+1], atoi(argv[i*2+2]));

   for (i=0; i<numIndividuos; i++)   // Impresión de los datos del array
     printf("Individuo %d: Nombre %s, Edad %d\n",  i, misDatos[i].nombre, misDatos[i].edad);

   free(misDatos);    // Liberar memoria
}
```

RECURSIVIDAD

1. Ordenación mediante método de la Burbuja recursivo: Crear una función llamada "**burbujaRecursiva**" que ordena un array de números enteros en orden ascendente con un algoritmo recursivo. Los números se introducen por teclado en la función **main()** y a después se llama a la función para ordenar el array. Finalmente el array ordenado se imprime en pantalla desde la función **main()**.
El algoritmo recursivo es el siguiente: *"Explorar el array desde el primer elemento comparando cada elemento con su siguiente ([0] con [1], [1] con [2], [2] con [3],…) hasta alcanzar el último elemento del array. Si no estan en orden ascendente correcto, se intercambian uno con el otro. Si estan en el orden correcto, no se intercambian. Al final del proceso, el número mayor del array se encontrará almacenado en la última posición más alta del array. A continuación repetir el mismo proceso con todos los elementos bajos del array excepto el último. La recursión termina cuando el número de elementos del array a ordenar es uno o cuando no se ha necesitado ningún intercambio en la última exploración del array"*.
La función **burbujaRecursiva** necesita dos argumentos de entrada: el nombre del array a ordenar y el número de elementos que contiene.

```
#define N 10                              //número de elementos del array
void BurbujaRecursiva(int *, int);    //prototipo de la función recursiva

main( )
{
  int num[N], i, r;
  system("cls");
  printf("Introduce los %d numeros del array:\n", N);
  for (i=0; i<N; i++)    //bucle para introducir los numeros del array desde el teclado
  {
     printf("Número %d: ", i+1); r=scanf("%d", &num[i]);
     if (r==0) i--;
  }
  BurbujaRecursiva(num, N);    //llamada a la función recursiva
  printf("Los numeros en orden ascendente son:\n");
  for (i=0; i<N; i++)
     printf("Número %d: %d\n", i+1, num[i]);
}
```

Solución:

```
void BurbujaRecursiva(int *array, int numelems)      //función recursiva
{ int i, aux, cambio=0;
   if (numelems==1) return;     //fin de la recursión
   for (i=0; i<numelems-1; i++)    //bucle de elementos a comparar en el array
   {
      if (array[i]>array[i+1])       //si los dos elementos estan desordenados
      {  aux=array[i]; array[i]=array[i+1]; array[i+1]=aux;      //intercambiar los dos elementos
         cambio=1;           //indica si se ha realizado un intercambio
      }
   }
   if (cambio)     //si ha habido algún intercambio
     BurbujaRecursiva(array, numelems-1);      //llamada recursiva
```

```
}
```

---

2. Realizar una función recursiva llamada **fact** que acepte un número entero **int** de entrada y devuelva el **factorial** de dicho número como valor de tipo real **double**. Si el número de entrada es negativo, deberá obtener el factorial de su inverso positivo.

Solución:

```
double fact(int num)
{
  if (num<0) num = -num;
  if (num==0) return 1.0;
  else return num * fact(num-1);
}
```

---

3. Programa para imprimir un número entero int en binario

Solución:

```
main()
{
 int num;
 printf("Dame un numero entero positivo: ");
 scanf("%d", &num);
 printf("El numero escrito en binario es: ");
 imprimebits(num);
}

void imprimebits(int n)
{
 if (n<2) printf(" %d", n);
 else { imprimebits(n/2);  printf(" %d", n%2); }
}
```

---

4. Diseñe un programa que calcule el **factorial** de un número entero entre **0 y 12** que se recibe por la línea de comandos, e imprima el resultado. La implementación del cálculo del número factorial debe realizarse mediante una función recursiva. La función **main()** debe comprobar que se han recibido correctamente los parámetros de la línea de comandos.

Solución:

```
long factorial(long n)
{
  if (n == 0) return 1;
  return (n*factorial(n-1));
}
void main(int argc, char **argv)
{
  int num;
  if (argc!=2)
  {
     printf ("Ejecute el programa aportando un parámetro entero\n");
     exit (-1);
  }
  num = atoi(argv[1]);
  if (num<0 || num>12)
  {
```

```
      printf ("El parámetro debe estar comprendido entre 0 y 12\n");
      exit (-1);
  }
  printf ("El factorial de %d es %ld\n", num, factorial(num));
}
```

FICHEROS

1. Diseñar una función con el siguiente prototipo **int fun(char *nom);**
La función debe mostrar por pantalla el contenido de un archivo cuyo nombre se pasa como argumento **nom** y tiene que devolver el número de caracteres que contiene. Suponer que el archivo se abre sin errores.

Solución:

```
int fun(char *nom)
{
  int con=0; char c;
  FILE *p=fopen(nom,"r");
  while ((c=fgetc(p))!=EOF)
    { con++; putchar(c); }
  fclose(p);
  return con;
 }

O BIEN:

int fun(char *nom)
{
  int con=0; char c;
  FILE *p=fopen(nom,"r");
  c=fgetc(p);
  while(!feof(p) && !ferror(p))
  { con++; printf("%c",c);  c=fgetc(p);  }
  fclose(p);
  return con;
 }
```

---

2. Crear un programa que lea del disco byte a byte un fichero de texto compuesto por varias frases y que cuente el número de frases, palabras y caracteres, mostrando finalmente estos resultados en pantalla. El nombre del fichero a leer se toma desde la linea de órdenes.
Consideramos que una frase terminará con un carácter **\n** o será la última frase en el fichero si no termina con **\n**. Dos frases consecutivas pueden estar separadas por más de un carácter **\n**.
Consideramos que una palabra termina con uno o más espacios en blanco o será la última palabra del fichero si no termina con blanco. Dos palabras consecutivas pueden estar separadas por más de un espacio en blanco.

Solución:

```
#include <stdio.h>
main(int argc, char *argv[ ])
{
  FILE *pf;
  long letras=0,palabras=0,frases=0;    // variables contadoras
  char car,carprev=' ';
  if (argc!=2)      //debe ser pasado un argumento en la linea de ordenes
    { printf("ERROR. Use este formato: CONTAR nombre-fichero"); exit(1); }
```

```
  if ((pf=fopen(argv[1],"r"))==NULL)    //abre el fichero de texto
    { printf("ERROR: Fichero %s no existe.",argv[1]); exit(1); }
  car = fgetc(pf);   //lee el primer character del fichero
  while (!feof(pf) && !ferror(pf))    //bucle de acceso secuencial
  {
    letras++;      //contador de caracteres
    switch (car)
    {
      case '\n': if (carprev!='\n') frases++;     //contador de frases
      case ' ':  if (carprev!='\n' && carprev!=' ') palabras++;     //contador de palabras
    }
    carprev = car; car = fgetc(pf);    //lee el siguiente caracter del fichero
  }
  if (ferror(pf))
    printf("ERROR al leer el fichero %s.",argv[1]);
  else
    { if (carprev!='\n')  frases++;    //si el ultimo character del fichero no era \n
      if (carprev!='\n' && carprev!=' ')  palabras++;   // si el ultimo character del fichero no era \n o  blanco
      printf("El fichero %s tiene:\n");
      printf("\t%ld caracteres.\n\t%ld palabras.\n\t%ld frases.", argv[1], letras, palabras, frases);
    }
  fclose(pf);
}
```

---

3. Escribir un programa que reciba desde la línea de órdenes, como único argumento, el nombre de un fichero de texto ya existente en el disco.
El programa debe mostrar por pantalla todo el contenido de dicho fichero de texto, pero convertido a minúsculas. El programa finalmente imprimirá también en pantalla el número total de caracteres que tal fichero contiene. El fichero debe ser abierto por el programa en modo lectura y finalmente cerrado. Si el fichero de texto no existe se emitirá un mensaje de error y el programa terminará.
Se puede utilizar la función estándar **tolower().**

Solución:

```
#include <stdio.h>
main (int argc, char *argv[ ])
{
   FILE *pf;
   char car;
   int cont=0;

   if (argc!=2) { printf("Argumentos de entrada incorrectos."); exit(1); }
   if ((pf=fopen(argv[1],"r"))==NULL)  { printf("El fichero no existe"); exit(1); }

   car = fgetc(pf);
   while (!feof(pf) && !ferror(pf))
   {
      printf("%c", tolower(car));
      cont++;
      car = fgetc(pf);
   }
   fclose(pf);
   printf(\nEl número de caracteres del fichero es: %d\n", cont);
}
```

---

4. Crear un programa que genere en disco un archivo de texto encriptado de nombre **"mifi.txt"** cuyo contenido se va introduciendo caracter a caracter desde el teclado. La encriptación consistirá en ir añadiendo +**1** al código ASCII de cada carácter (A se convierte en B, B se convierte en C,...), para así conseguir que

alguien que lea accidentalmente el archivo no lo pueda entender. La introducción del texto finalizará cuando se pulse la marca de Fin de Archivo del teclado (Ctrl+Z o Ctrl+D).

Solución:

```
main()
{
  char c;
  FILE *pf;
  pf=fopen("mifi.txt","w");
  if (pf==NULL)  { printf("Error de acceso a fichero.\n"); exit(0); }
  printf("Introduce texto terminando con Ctrl+Z: ");
  while((c=getchar())!=EOF && !ferror(pf))
     fputc(c+1, pf);
  if (ferror(pf))  printf("Error al grabar el fichero.\n");
  fclose(pf);
}
```

Otra solución:

```
main()
{
  char c;
  FILE *pf;
  pf=fopen("mifi.txt","w");
  if (pf==NULL)  { printf("Error de acceso a fichero.\n"); exit(0); }
  printf("Introduce texto terminando con Ctrl+Z: ");
  c = getche();
  while(!feof(stdin) && !ferror(pf))
     {  fputc(c+1, pf); c = getche();  }
  if (ferror(pf))  printf("Error al grabar el fichero.\n");
  fclose(pf);
}
```

---

5. Escribir un programa que cree un fichero de texto cuyo nombre se pide inicialmente al ejecutar el programa y cuyo contenido se obtenga por teclado carácter a carácter. La introducción del texto a través del teclado debe acabar al introducir el carácter **$**.

Solución:

```
main()
{
  char c, nombre[20];
  FILE *pf;
  printf("Dame nombre del fichero: ");  gets(nombre);
  pf=fopen(nombre,"w");
  if (pf==NULL)  { printf("Error de acceso a fichero.\n"); exit(0); }
  printf("Introduce texto terminando con $ y Enter: ");
  while((c=getchar())!=´$´ && !ferror(pf))
     fputc(c, pf);
  if (ferror(pf))  printf("Error al grabar el fichero.\n");
  fclose(pf);
}
```

Otra solución:

```
main()
{
  char c, nombre[20];  FILE *pf;
```

```
  printf("Dame nombre del fichero: ");  fgets(nombre,20,stdin);
  if (nombre[strlen(nombre)-1]=='\n')  nombre[strlen(nombre)-1]='\0';
  pf=fopen(nombre,"w");
  if (pf==NULL)  { printf("Error de acceso a fichero.\n"); exit(0); }
  printf("Introduce texto terminando con $: ");
  c = getche();
  while(c!=´$´ && !ferror(pf))
    {  fputc(c, pf); c = getche();  }
  if (ferror(pf))  printf("Error al grabar el fichero.\n");
  fclose(pf);
}
```

---

6. Diseñar una función con el siguiente prototipo: **int fun(FILE *pf);** donde el argumento **pf** es el puntero al fichero **pru** que se ha creado en el programa principal. La función ha de visualizar el contenido del fichero **pru** y además tiene que devolver el número de líneas que han sido leídas del fichero. En la función **pru** se descartarán las posibles líneas intermedias en blanco, que no se visualizarán ni se contabilizarán.
El programa principal es:

```
int fun(FILE *pf);
main()
{
  int n;
  FILE *pf;
  char cad[80];
  if ((pf=fopen("pru", "w+"))==NULL)  { printf("Error de apertura\n"); exit(0); }
  puts("Dame un texto (CTRL+Z para terminar): ");
  while(gets(cad)!=NULL)
  { fputs(cad, pf); fputc('\n',pf);
    puts("Dame un texto (CTRL+Z para terminar): ");
  }
  rewind(pf);
  n=fun(pf);
  printf("Numero de cadenas=%d\n",n);
  fclose(pf);
}
```

Solución:

```
int fun(FILE *pf)
{ char cad[80];
   int n=0;
   fgets(cad,80,pf);
   while(!feof(pf) && !ferror(pf))
   {  if (cad[0]!='\n')
      {  printf("%s\n",cad); n++;  }
      fgets(cad,80,pf);
   }
   return (n);
}
```

---

7. Realizar un programa que cree un fichero de texto cuyo nombre es el primer argumento de la línea de órdenes y cuyo contenido son una serie de líneas separadas por caracteres **\n** que contienen cada una el resto de los argumentos de la línea de órdenes, escritos en mayúsculas.
Por ejemplo, la orden: **prog mifi juan paco pepe**
crea el archivo de nombre **mifi** que contiene las líneas:

JUAN
PACO
PEPE

Solución:

```
int main(int argc, char *argv[ ])
{  int i,j;
   FILE *pf=fopen(argv[0],"w+");
   if (pf==NULL)  { printf("Error de creación del fichero.\n"); exit(0); }
   for (i=1; i<argc; i++)
   {  for (j=0; argv[i][j]!=0; j++)
         if (argv[i][j]>='a' && argv[i][j]<='z')
            argv[i][j]+=('A'-'a');
      fputs(argv[i], pf); fputc('\n', pf);
   }
   fclose(pf);
   printf("El fichero %s ha sido creado.\n", argv[0]);
}
```

---

8. Escribir un programa que realice lo siguiente:
-1º: Graba un fichero **salida.txt** que contiene los argumentos que se dan en la linea de ordenes, uno por linea. No se considera argumento de entrada el propio nombre del programa ejecutable.
-2º: Lee el fichero **salida.txt** y presenta su contenido en la pantalla.

Solución:

```
main (int argc,char *argv[ ])
{
  FILE *pfile;
  char cadena[100];
  int i=0;
  if ((pfile=fopen("salida.txt","w"))==NULL)
  { puts("\nNo se puede escribir el fichero\n"); exit(0); }


  // 1º. GRABAR FICHERO salida.txt
  // OPCION UNO (con indices)
  for (i=1; i<argc; i++)
  { fputs(argv[i],pfile); fputs("\n",pfile); }

  // OPCION DOS (con punteros)
  while (*++argv)
  { fputs(*argv,pfile); fputs("\n",pfile); }


  // 2º. LEER FICHERO salida.txt
  fclose(pfile);
  if ((pfile=fopen("salida.txt","r"))==NULL)
  { puts("\nNo se puede leer el fichero\n"); exit(0); }
  while(fgets(cadena,100,pfile)!=NULL)
    printf("%s",cadena);
  fclose(pfile);
}
```

---

9. Crear un programa que acepte por teclado sucesivas parejas de datos Nombre (40 bytes máximo) y DNI (10 bytes máximno) de varias personas y las almacene en un fichero de texto llamado "datos.txt", separados todos ellos por caracteres "nueva linea" (\n). La introducción terminará cuando el nombre de una nueva persona se deje vacío.
El Nombre debe ser introducido en formato "Apellidos, Nombre" (con una coma). Si es introducido en formato "Nombre + Apellidos" (sin coma), el programa presentará un mensaje de error y pedirá una nueva introducción del dato correcto.

El DNI debe tener 8 dígitos y una letra final. En caso contrario, el programa presentará un mensaje de error y pedirá una nueva introducción del dato correcto.
Finalmente, el programa mostrará en la pantalla todos los datos almacenados en el fichero y lo cerrará. Una nueva ejecución del prorama no destruirá los datos ya almacenados en previas sesiones, y los nuevos datos serán añadidos al final.

Solución:

```
#include <stdio.h>
int oknombre(char *cad)
{
  int i=0, cuenta=0;
  while (cad[i]!=0)
    if (cad[i]==',')  cuenta++;
  if (cuenta!=1)  { printf("Error, el formato es:  Apellidos, Nombre\n"); return 0; }
  return 1;
}

int okdni(char *cad)
{
  int i=0;
  while (cad[i]>='0' && cad[i]<='9') i++;
  if (i!=8)  { printf("Error en el número de dígitos\n"); return 0; }
  if (cad[8]==0) { printf("Error: falta la letra final\n"); return 0; }
  if (cad[8]<'A' || cad[8]>'Z') { printf("Error en la letra final\n"); return 0; }
  return 1;
}

main( )
{
 FILE *pf;
 char *nomfile="datos.txt", nombre[40], dni[10];

 if ((pf=fopen(nomfile,"a+"))==NULL)
   {  printf("Error al abrir el fichero\n"); exit(0); }
 printf("Dame nombres y DNIs (vacío = salir):\n\n");
 while(!ferror(pf))
 {
   do
   {  printf("Nombre: "); gets(nombre); if (nombre[0]==0) break;
   } while (!oknombre(nombre));
   if (nombre[0]==0) break;
   do
   {  printf("DNI: "); gets(dni);
   }  while (!okdni(dni));
   fputs(nombre,pf); fputc('\n',pf); fputs(dni,pf); fputc('\n',pf);
   printf("\n");
 }

 printf("Los datos en el fichero son los siguientes:\n\n");
 rewind(pf);
 fgets(nombre,40,pf); fgets(dni,10,pf);
 do while (!feof(pf) && !ferror(pf))
 {
   printf("Nombre: %sDNI: %s\n",nombre,dni);
   fgets(nombre,40,pf); fgets(dni,10,pf);
 }
 fclose(pf);
}
```

---

10. Programa para ordenar alfabéticamente un fichero de texto compuesto por sucesivos nombres de persona (Apellidos, Nombre) separados por caracteres "nueva linea" (**\n**).

Solución:

```
#include <stdio.h>
main()
{
  char nom[20], cad1[40], cad2[40];
  FILE *pf; int cambio=1; long posic;
  printf("Dame nombre del fichero a ordenar: "); gets(nom);
  if ((pf=fopen(nom,"r+"))==NULL)  { printf("ERROR: fichero no existente."); exit(0); }
  while (cambio && !ferror(pf))
  {  cambio=0; rewind(pf);
     fgets(cad1,40,pf); fgets(cad2,40,pf);
     while (!feof(pf) && !ferror(pf))
     {  if (strcmp(cad1,cad2)>0)
        { posic=ftell(pf);
          fseek(pf,-strlen(cad1)-strlen(cad2)-2,SEEK_CUR); fputs(cad2,pf); fputs(cad1,pf);
          fseek(pf,posic,SEEK_SET); cambio=1;
        }
        else strcpy(cad1,cad2);
        fgets(cad2,40,pf);
     }
  }
  if (ferror(pf))  printf("Error de acceso a fichero.");
  else printf("El fichero ha sido ordenado.");
  fclose(pf);
}
```

---

11. Programa para fusión de dos ficheros de texto con nombres ordenados alfabéticamente: Crear un programa que solicite por teclado el nombre de dos ficheros de texto de entrada ya existentes en el disco, que contienen nombres de personas ordenados alfabéticamente separados por caracteres **\n**, y pida también por teclado un tercer nombre de un fichero de salida que será creado con la fusión de ambos ficheros de entrada, obteniendo una lista total de nombres también ordenados alfabéticamente y separados con caracteres **\n**.

Solución:

```
#include <stdio.h>
main()
{
  char nom1[20], nom2[20], nom3[20], cad1[40], cad2[40];
  FILE *pf1, *pf2, *pf3;
  printf("Dame nombre del primer fichero de entrada: "); gets(nom1);
  if ((pf1=fopen(nom1,"r"))==NULL)
    { printf("ERROR: fichero no existente."); exit(0); }
  printf("Dame nombre del segundo fichero de entrada: "); gets(nom2);
  if ((pf2=fopen(nom2,"r"))==NULL)
    { printf("ERROR: fichero no existente."); fclose(pf1); exit(0); }
  printf("Dame nombre del fichero de salida: "); gets(nom3);
  if ((pf3=fopen(nom3,"w"))==NULL)
    { printf("ERROR: fichero no puede abrirse."); fclose(pf1); fclose(pf2); exit(0); }
  fgets(cad1,40,pf1); fgets(cad2,40,pf2);
  while (!(feof(pf1) && feof(pf2)))
  {  if (feof(pf1))
       { fputs(cad2,pf3); fgets(cad2,40,pf2); }
     else if (feof(pf2))
```

```
        { fputs(cad1,pf3); fgets(cad1,40,pf1); }
      else if (strcmp(cad1,cad2)<0)
        { fputs(cad1,pf3); fgets(cad1,40,pf1); }
      else
        { fputs(cad2,pf3); fgets(cad2,40,pf2); }
    }
    fclose(pf1); fclose(pf2); fclose(pf3);
    printf("\nEl fichero %s ha sido creado.", nom3);
  }
```

---

12. Se ha creado un fichero usando la función **fwrite()**, que almacena estructuras del siguiente tipo:

```
struct pp
{ char nom[10];
  int edad;
};
```

Suponiendo que el fichero ya está abierto para lectura y es referenciado por el puntero **pun**, indique cómo debe ser la función **fun()** para que pregunte por teclado qué número de registro se quiere leer (entendiendo que el primer registro del fichero es el Nº 1) y devuelva como resultado dicho registro completo.

```
struct pp fun(FILE *pun)
{
    /* Instrucciones  */
}
```

Solución:

```
struct pp fun(FILE *pun)
{
  struct pp una={"",0};  int n;
  printf("Numero de registro a leer: ");  scanf("%d",&n);
  fseek(pun,sizeof(struct pp)*(n-1),SEEK_SET);
  fread(&una,sizeof(struct pp),1,pun);
  if (feof(pun)) printf("Registro no existe.\n");
  return una;
}
```

---

13. Escribir una función llamada **fun** con el prototipo indicado, que escriba carácter a carácter la cadena de caracteres apuntada por el puntero **cad**, sobre el fichero abierto en modo escritura y apuntado por el puntero **pf**, sin utilizar ninguna variable adicional aparte de **cad** y **pf**.

```
void fun(char *cad, FILE *pf);
```

Solución:

```
void fun(char *cad, FILE *pf)
{
   while (*cad!=0 && !ferror(pf))
   {  fputc(*cad, pf); cad++;  }
}
```

---

14. Escribir un programa que pida por teclado el nombre de un fichero de texto y, si el fichero existe, lea caracter a caracter su contenido e imprima en pantalla el número de vocales que contiene. Si el fichero no existe, imprimir un mensaje que lo advierta. Suponer que no existen vocales acentuadas, y sí pueden existir mayúsculas y minúsculas.

Solución:

```
#include <stdio.h>
#include <string.h>
main()
```

```
{
  char nomfi[20], car; int num=0;
  FILE *pf;
  printf("Dame nombre del fichero: ");  gets(nomfi);
  if ((pf=fopen(nomfi,"r+"))==NULL)
  {  printf("El fichero %s no existe.\n", nomfi); exit(0);  }
  car=tolower(fgetc(pf));
  while (!feof(pf) && !ferror(pf))
  {  if (car=='a' || car=='e' || car=='i' || car=='o' || car=='u')  num++;
     car=tolower(fgetc(pf));
  }
  printf("Hay %d vocales en el fichero.\n", num);
  fclose(pf);
}
```

---

15. Existe en el disco un fichero de nombre **datos.dat** que contiene registros desordenados de tipo **registro**. Escribir un programa que obtenga a partir de él otro fichero llamado **datosord.dat** que contenga los mismos registros, pero ordenados ascendentemente por el campo **clave**. El fichero original **datos.dat** no debe ser modificado. Los pasos a seguir son los siguientes:
1º. Copiar el fichero **datos.dat** en **datosord.dat**.
2º. Aplicar algún método de ordenación de registros al fichero **datosord.dat**.

```
typedef struct reg
{  int clave;  char nombre[40];  } registro;
```

Solución-1:

```
main()
{  registro dato, dato2;   FILE *pf1, *pf2;
  int i, j, numregs=0, tamreg=sizeof(registro);
  //1º
  if ((pf1=fopen("datos.dat", "r+"))==NULL)
    {  printf("Fichero datos.dat no existe\n"); exit(0);  }
  if ((pf2=fopen("datosord.dat", "w+"))==NULL)   {  printf("Error de apertura\n"); exit(0);  }
  while (fread(&dato, tamreg, 1, pf1)==1)
    {  fwrite(&dato, tamreg, 1, pf2); numregs++;  }
  fclose(pf1);
  //2º
  for (i=0; i<numregs-1; i++)    // Aplicar método de la burbuja
     for (j=i+1; j<numregs; j++)
     {  fseek(pf2, i*tamreg, SEEK_SET); fread(&dato, tamreg, 1, pf2);
        fseek(pf2, j*tamreg, SEEK_SET); fread(&dato2, tamreg, 1, pf2);
        if (dato.clave > dato2.clave)
        {  fseek(pf2, i*tamreg, SEEK_SET); fwrite(&dato2, tamreg, 1, pf2);
           fseek(pf2, j*tamreg, SEEK_SET); fwrite(&dato1, tamreg, 1, pf2);
        }
     }
  fclose(pf2);
}
```

Solución-2:

```
main()
{  registro dato, dato2;   FILE *pf;
  int i, numregs=0, tamreg=sizeof(registro), cambio=1;
  //1º
  system("cp datos.dat datosord.dat");
  if ((pf=fopen("datosord.dat", "r+"))==NULL)
  {  printf("Fichero datos.dat no existe\n"); exit(0);  }
  fseek(pf, 0, SEEK_END);  numregs = ftell(pf)/tamreg;
```

```
  //2º
  while (cambio)
  {  for (i=cambio=0; i<numregs-1; i++)
     {  fseek(pf, i*tamreg, SEEK_SET);
        fread(&dato, tamreg, 1, pf);  fread(&dato2, tamreg, 1, pf);
        if (dato.clave > dato2.clave)
        {  fseek(pf, i*tamreg, SEEK_SET);
           fwrite(&dato2, tamreg, 1, pf);  fwrite(&dato, tamreg, 1, pf); cambio=1;
        }
     }
  }
  fclose(pf);
}
```

---

16. Crear un programa que abra un fichero de texto de nombre "**fich.txt**" (el fichero debe existir), lo lea letra a letra y, a medida que lo va leyendo, convierta a mayúscula la primera letra de cada palabra del texto y la grabe en su lugar en el fichero. Al final, el fichero quedará grabado con todas sus palabras comenzando por letra mayúscula. Se puede utilizar la función **toupper()**.
Algoritmo sugerido: Leer cada letra del fichero en una variable **car**, y guardar la anteriormente leída en **carprev**. Si la letra recién leída es la primera de una palabra, retroceder el apuntador de Lectura/Escritura un byte, y grabar ahí la letra convertida a mayúsculas. Una letra es la primera de una palabra cuando viene precedida por un espacio en blanco, por un carácter '\n', o cuando es la primera de todo el fichero.

Solución:

```
main()
{
  char car, carprev=' ';
  FILE *pf = fopen("fich.txt", "r+");
  if (pf==NULL)
    {  printf("ERROR: fichero fich.txt no existe\n"); exit(0);  }

  car = fgetc(pf);
  while (!ferror(pf) && !feof(pf))
  {  if (carprev==' ' || carprev=='\n')
     {  fseek(pf, -1, SEEK_CUR); fputc(toupper(car), pf);  }
     carprev = car; car = fgetc(pf);
  }

  if (ferror(pf))
     printf("ERROR de acceso al fichero fich.txt\n");
  else
     printf("El fichero fich.txt ha sido modificado correctamente\n");
  fclose(pf);
}
```

LISTAS ENLAZADAS

1. Una lista enlazada está formada por estructuras con la siguiente definición:

```
struct pp
{  int n;
   struct pp *sig;
};
```

Diseñar la función cuyo prototipo es **void fun(struct pp *p);** que debe eliminar el elemento de la lista enlazada apuntado por el puntero **p**.

Solución:

```
void fun(struct pp *p)
{
  struct pp *aux;
  aux=p->sig;  *p=*aux;
  free(aux);
}
```

---

2. Una lista enlazada está formada por estructuras del tipo: **struct pp**
{ int dato; struct pp *sig; };

El puntero que apunta al inicio de la lista es la variable global **pini**, distinto de NULL (la lista ya tiene datos cargados).
Escribir la función con prototipo: **int existe_dato(int x);** que retorna el siguiente valor:

- **0** si el dato **x no existe** en la lista
- **1** si el dato **x** existe y es **el primero** de la lista (el apuntado por **pini**).
- **2** si el dato **x** existe y **no es el primero ni el último** de la lista.
- **3** si el dato **x** existe y **es el último** de la lista (el que contiene NULL en su campo **sig**).

Si la lista sólo contiene un elemento, la función lo considerará como si fuera el primer elemento.

Solución:

```
int existe_dato(int x)
{
  int resul=0;
  struct pp *q=pini;
  while (q!=NULL && resul==0)
  {  if (q->dato==x)
     { if (q==pini) resul=1;
       else if (q->sig==NULL) resul=3
       else resul=2;
     }
     q = q->sig;
  }
  return resul;
}
```

---

3. Una lista enlazada está formada por estructuras del tipo **nodo:**

```
typedef struct nodo
{ int dato;
  struct nodo *sig;
} nodo;
```

Escribir la función **float fun(nodo *p)** que calcula la media aritmética de todos los campos **dato** de las estructuras de la lista. La función **fun** recibe en el parametro **p** la dirección inicial de la lista, y tiene que devolver la media aritmética.

Solución:

```
float fun (nodo * p)
{
  float suma = 0;  int cont = 0;
  while (p != NULL)
  {   suma += p->dato; cont ++; p = p->sig;  }
  return (suma/cont);
}
```

---

4. Una **lista doblemente enlazada** con estructuras del tipo:

```
typedef struct lista
{ int dato;
  struct lista *panterior, *psiguiente;
} ldoble;
```

está referenciada por dos variables globales, punteros de **tipo ldoble** llamados **pini** y **pfin** que apuntan al comienzo y al final de la lista respectivamente. Si la lista está vacía, ambos valen NULL.
Escribir la funcion de prototipo: **void anadir(int dato, int final);**
que añade el valor **dato** a la lista, y lo añade antes del primer elemento de la misma (el apuntado por **pini**) si el argumento **final** vale **0**, o bien después del elemento final de la lista (el apuntado por **pfin**) si el argumento **final** vale distinto de 0. La función también debe operar bien para el caso de que la lista esté inicialmente vacía.

Solución:

```
void anadir(int dato, int final)
{
  ldoble *q = (ldoble *)malloc(sizeof(ldoble));
  if (q==NULL) { puts("\nNo hay memoria\n"); exit(0); }
  q->dato=dato;
  if (pini==NULL)        // Lista vacia
     { q->pant=q->psig=NULL; pini=pfin=q; }
  else if (final)        // Se añade al final
     { q->psig=NULL; q->pant=pfin; pfin->psig=q; pfin=q; }
  else                   // Se añade al principio
     { q->psig=pini; q->pant=NULL; pini->pant=q; pini=q; }
}
```

---

5. Una lista enlazada está formada por estructuras de tipo:

```
struct pp
{ int dato;
  struct pp *sig; };
```

La lista está referenciada por la variable global **pini**, puntero al inicio de la lista. Escribir la función con prototipo: **int existeDato(int dd);** que retorna el siguiente valor numérico:

- **0** si el dato **dd** no existe en la lista.
- **1** si el dato **dd** es el primero de la lista o es el único existente en ella.
- **2** si el dato **dd** no es el primero ni el último de la lista.
- **3** si el dato **dd** es el último de la lista.

Solución:

```
int existeDato(int dd)
{
  struct pp *paux=pini;
  if (pini==NULL) return 0;          // No hay lista, dato dd no existe
  if (pini->dato==dd) return 1;      // Es el primero
  while (paux!=NULL)
  {
    if (paux->dato==dd)              // Dato dd existe
    {
      if (paux->sig==NULL) return 3;  // Es el ultimo
      else return 2;                  // No es el ultimo
    }
    paux=paux->sig;
  }
  return 0;                          // No existe
}
```

---

6. Crear una función **InsertAnt** con el prototipo indicado, que sirva para insertar en una lista doblemente enlazada formada por estructuras del tipo **struct s**, un nuevo valor **x** antes del elemento apuntado por **p**. Este elemento apuntado por **p** puede ser incluso el primer elemento de la lista. **struct s { int num; struct s *anterior, *siguiente; }**

void insertAnt(int x, struct s *p); <= Prototipo de la función

Solución:

```
void insertAnt(int x, struct s *p)
{
  struct s *ant=p->anterior;
  struct s *q=(struct s *)malloc(sizeof(struct s));
  if (q==NULL)
     { printf("Memoria llena.\n"); return; }
  q->num=x; q->siguiente=p; q->anterior=ant; p->anterior=q;
  if (ant!=NULL)
     ant->siguiente=q;
}
```

Otra solución:

```
void insertAnt(int x, struct s *p)
{
  struct s *q=(struct s *)malloc(sizeof(struct s));
  if (q==NULL)
     { printf("Memoria llena.\n"); return; }
  *q = *p;  p->num=x;
  p->siguiente->anterior=q; p->siguiente=q; q->anterior=p;
}
```

---

7. Escribir un programa completo que tome desde teclado sucesivos números enteros y los almacene en una **lista enlazada simple**, gobernada por un puntero ***lista**, insertando cada nuevo número por el final de la lista (después del elemento que contiene un valor NULL como puntero al siguiente elemento). La introducción de datos terminará cuando se pulse **CTRL+D** (marca de fin de fichero del teclado). Los elementos de la lista contendrán estructuras del siguiente tipo:

```
struct elemento
{  int dato;
   struct elemento *psig;  }
```

Una vez terminada la introducción de datos, el programa los mostrará todos en pantalla y finalmente eliminará de la memoria toda la lista.

Solución:

```
#include <stdio.h>
struct elemento
{  int dato;
   struct elemento *psig;
}

main()
{
   struct elemento *lista=NULL, *q, *p;
   int num;
   while (1)
   {  printf("Dame número (CTRL+D=Fin): ");
      if (scanf("%d", &num)==EOF) break;
      q = (struct elemento *)malloc(sizeof(struct elemento));
      if (q==NULL) { printf("ERROR: Falta memoria.\n"); getchar(); }
      else
      {  q->dato=num; q->psig=NULL;
```

```
      if (lista==NULL) lista=q;
      else
      {  p=lista;
         while (p->psig!=NULL)  p=p->psig;
         p->psig=q;
      }
    }
  }

  printf("\nLos números introducidos son:\n");
  q=lista;
  while (q!=NULL)
  { printf("%d   ",q->dato); q=q->psig; }

  printf("\nEliminando la lista...\n");
  while (lista!=NULL)
  { q=lista; lista=lista->psig; free(q); }
}
```

---

8. Una **lista doblemente enlazada** con estructuras del tipo:

```
typedef struct cole
{ int dato;
  struct cole *pant;
  struct cole *psig;
} ldoble;
```

está referenciada por dos variables globales, punteros de **tipo ldoble** llamados **pini** y **pfin** que apuntan al comienzo y al final de la lista. Si la lista está vacía, ambos valen NULL.
Escribir la funcion de prototipo: **void anadir(int dato);**
que añade un dato **int** a la lista, bien antes del primer elemento de la misma (el apuntado por **pini**) o bien después del elemento final de la misma (el apuntado por **pfin**), a elección del alumno (indicar qué elección es la que se ha tomado).

Solucion:

```
void anadir(int dato)
{
  ldoble *q;
  q=(ldoble *)malloc(sizeof(ldoble));
  if (q==NULL) { puts("\nNo hay memoria\n"); exit(0); }
  q->dato=dato;
  if (pini==NULL)          // Lista vacia
  { q->pant=q->psig=NULL; pini=pfin=q; }

  //-------------------------------------------
  else                     // Opcion 1: se añade al inicio de la lista
  { q->psig=pini; q->pant=NULL; pini->pant=q; pini=q; }

  //-------------------------------------------
  else                     // Opcion 2: se añade al final de la lista
  { q->psig=NULL; q->pant=pfin; pfin->psig=q; pfin=q; }
}
```

---

9. Dado el siguiente programa que rellena una lista simplemente enlazada añadiendo los elementos nuevos por el principio (elementos tomados del **array[]**), la muestra en pantalla y finalmente la libera eliminando los elementos por el final:

- Escribir el código para las funciones siguientes:
  void insertarAlPrincipio(node_t **p_cab, int num)

void mostrarLista(nodo_t *p_cab)
void eliminarAlFinal(nodo_t **pp_cab)

- Sustituir los COMENTARIOS por las llamadas a las funciones de insertar, mostrar y eliminar en el siguiente programa main.

```
struct nodo
{
   int numero;
   struct nodo *p_next;
};
typedef struct nodo nodo_t;

int main()
{
   int array[] = {39, 65, 85, 15, 8, 5, 93, 87, 50, 55, 69, 6, 40, 16, 25, 49, 100, 38, 86, 1};
   unsigned int i = 0;

   nodo_t *p_cab = NULL;   // Puntero a la cabecera de la lista

   printf("Guardar en la lista\n"); printf("----------\n" );

   for (i=0; i<sizeof(array)/sizeof(int); i++)
   {
         /* COMENTARIO 1 */
   }

   printf("La lista guardada es\n");
   printf("----------\n" );

   /* COMENTARIO 2 */

   printf("Liberamos la lista\n"); printf("----------\n" );
   while (p_cab!=NULL)
   {
         /* COMENTARIO 3 */
   }
}
```

Solución:

```
/* COMENTARIO 1 */
insertarAlPrincipio(&p_cab, array[i]);

/* COMENTARIO 2 */
mostrarLista(p_cab);

/* COMENTARIO 3 */
eliminarAlFinal(&p_cab);

void insertarAlPrincipio(node_t **p_cab, int num)
{       if (p_cab!=NULL)
        {   // Create a new element
            nodo_t *p_new = (nodo_t *) malloc(sizeof(nodo_t));
            if (p_new != NULL)
            {    // Fill the new element
                 p_new->numero = num;  p_new->p_next = *p_cab;

                 // Point the header to the new element
                 *p_cab = p_new;
```

```
            }
        }
}

void mostrarLista(nodo_t *p_cab)
{   if (p_cab==NULL)
    {   printf("La lista está vacia\n");  }
    else
    {   nodo_t *p_aux=NULL;
        p_aux = p_cab;
        printf("Contenido de la lista:\n");
        while(p_aux != NULL)
        {   printf("%d  ", p_aux->numero);
            p_aux = p_aux->p_next;
        }
    }
}

void eliminarAlFinal(nodo_t **pp_cab)
{   if (pp_cab!=NULL)
    {
        //1 - Comprobar que la lista no esta vacia
        if (*pp_cab != NULL)
        {
            //2 - Recorrer la lista hasta encontrar el nodo último
            nodo_t *p_aux = *pp_cab;   nodo_t *p_ant = *pp_cab;
            while (p_aux->p_next!=NULL)
            {   p_ant = p_aux;
                p_aux = p_aux->p_next;
            }

            if (p_ant!=p_aux)
            {   //3 - Apuntar el penultimo nodo a NULL
                p_ant->p_next = NULL;
            }
            else //CASO ESPECIAL: Solo habia un nodo
            {   *pp_cab=NULL;  }

            //4 - Liberar el ultimo nodo
            free(p_aux);
        }
    }
}
```

ÁRBOLES BINARIOS

1. Existe en el disco un fichero de nombre **datos.dat** que contiene registros desordenados de tipo **registro** en los que el campo **clave** no se repite. Completar los COMENTARIOS del siguiente programa para que cree un árbol binario ordenado a partir de los registros del fichero **datos.dat**, con nodos que contengan todos los registros del fichero. Después se imprimirán en pantalla todos los registros por orden ascendente de **clave**, y finalmente el árbol se borrará de la memoria.

```
typedef struct reg
{
  int clave;
  char nombre[40];
} registro;
```

```
struct nodo
{
  registro datosreg;
  struct nodo *dcho, *izqdo;
};

struct nodo *anadir(registro dd, struct nodo *r)  // Función recursiva que crea el árbol
{
   if (r==NULL)
   {
     r = (struct nodo *)malloc(sizeof(struct nodo));
     if (r!=NULL)
     {   r->datosreg = dd;
         r->dcho = r->izqdo = NULL;
     }
   }
   else
   {
     if (dd.clave < r->datosreg.clave)
         r->izqdo = anadir(dd, r->izqdo);
     else
         r->dcho = anadir(dd, r->dcho);
   }
   return r;
}

void imprimir(struct nodo *r)    // Función recursiva para imprimir por orden de clave
{  */ COMENTARIO-1 */    }

void borrararbol(struct nodo *r)  // Función recursiva para borrar el árbol:
{   /* COMENTARIO-2 */   }

main()
{
  registro  dato;  FILE *pf;
  struct nodo *raíz=NULL;
  // Instrucciones para abrir el fichero, leer todos sus registros e insertarlos en el árbol:
  /* COMENTARIO-3 */

  imprimir(raiz);
  borrararbol(raiz);
}
```

Solución:

```
*/ COMENTARIO-1 */
  if (r!=NULL)
  {  imprimir(r->izqdo);
     printf("Clave: %d, Nombre: %s\n", r->datosreg.clave, r->datosreg.nombre);
     imprimir(r->dcho);
  }

/* COMENTARIO-2 */
  if (r!=NULL)
  {  borrararbol(r->izqdo); borrararbol(r->dcho);  free(r);  }

/* COMENTARIO-3 */
  if ((pf=fopen("datos.dat", "r+"))==NULL)
  {  printf("Fichero datos.dat no existe\n"); exit(0);  }
```

```
while (fread(&dato, sizeof(registro), 1, pf)==1)
{
  raiz = anadir(dato, raiz);
}
fclose(pf);
```

ALGORITMOS DE ORDENACIÓN DE DATOS

1. Método Quicksort iterativo (con la ayuda de una pila):

```
#include <stdio.h>
#define N 10          //Número de elementos del array
void qsortiterativo(int *lista, int numelem);          //declaración prototipo

main()
{ int a[N]={10,3,7,5,12,1,27,3,8,13};  //array a ordenar
  int i;
  qsortiterativo(a, N);               //llamada a la función de ordenación
  printf("El array esta ordenado: ");
  for (i=0; i<N; i++) printf("%d ",a[i]);
}

void qsortiterativo(int *lista, int numelem)
{
   int posinf=0, possup=numelem-1, izq, der, mitad, aux, top=-1;
   int *pila = (int *)malloc(numelem);      // Crear una pila auxiliar con un array dinámico
   pila[++top]=posinf;                  // insertar en la pila valores iniciales de posinf y possup
   pila[++top]=possup;
   while (top>=0)                          // Seguir extrayendo de la pila mientras no esté vacía
   {   possup = pila[top--]; posinf = pila[top--];         // Extraer de la pila possup y posinf
      izq=posinf;       //izq comienza desde el extremo inferior del array
      der=possup;    //der comienza desde el extremo superior del array
      mitad = lista[(izq+der)/2];    //valor del elemento mitad del array (pivote)
      do
      {  while (lista[izq]<mitad && izq<possup) izq++;
        while (lista[der]>mitad && der>posinf) der--;
        if (izq<=der)
        { if (izq!=der)  { aux=lista[izq]; lista[izq]=lista[der]; lista[der]=aux; }
          izq++; der--;
        }
      } while (izq<=der);  //continuar búsqueda hasta que izq>der
      if (der>posinf)    // Si hay elementos a la izqda del pivote, insertar en la pila la parte izqda
        {  pila[++top] = posinf;  pila[++top] = der;  }
      if (izq<possup)    // Si hay elementos a la dcha del pivote, insertar en la pila la parte dcha
        {  pila[++top] = izq;  pila[++top] = possup;  }
   }
   free(pila);
}
```