```c
// Acceso a los parametros suministrados desde linea de comandos
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;

  printf("Se han recibido %d parámetros.\n", argc);
  for (i=0; i<argc; i++)
    printf("  Parámetro %d: |%s|\n", i, argv[i]);
  return 1;
}
```

```c
// Uso de la salida estandar de error
// Uso de la variable errno y de la funcion perror
// Este programa intenta abrir el fichero que se le pasa como argumento,
// y si no lo consigue informa del error
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  FILE *f;

  printf("Ejecutando programa...\n");
  if (argc<2) { // Si no se especifica argumento
    fprintf(stderr, "Uso: %s fichero\n", argv[0]);
    exit(-1);
  }
  printf("Abriendo fichero..\n");
  f=fopen(argv[1], "r");
  if (f==NULL) {
    perror("Al abrir fichero");
    fprintf(stderr, "Error número %d\n", errno);
    exit(-1);
```

```
24    }
25    printf("Fichero abierto con éxito\n");
26    fclose(f);
27    return 0;
28  }
```

```c
// Ejemplo de escritura de fichero usando el tipo FILE*
//
// Cuando el programa se invoca sin argumentos, este crea
// el fichero "Datos.txt" con salida formateada en ASCII
// y el fichero "Datos.dat" con datos binarios.
//
// Cuando se le llama con un argumento (su valor se ignora), los
// ficheros anteriores son leidos (en modo texto o binario) y los
// valores obtenidos son mostrados.
//
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  FILE *f_texto;  // Para el fichero formateado ASCII
  FILE *f_data;   // Para el fichero binario
  int x=10;
  int y=47;
  float g=1.0;
  char txt[5]="Hola"; // 5 = 4 letras mas terminador

  if (argc<2) { // No hay argumento, modo "crear ficheros"
    // Abrir fichero de texto
```

```
24      f_texto=fopen("Datos.txt", "w");
25      if (f_texto==NULL) {
26        perror("Creando Datos.txt"); exit(-1);
27      }
28
29      // Abrir fichero binario. La letra "b" en el modo de apertura se
30      // ignora bajo UNIX, pero tiene su importancia bajo DOS/Windows
31      f_data=fopen("Datos.dat", "wb");
32      if (f_data==NULL) {
33        perror("Creando Datos.dat"); exit(-1);
34      }
35
36      // Escribimos los datos como ASCII en Datos.txt
37      fprintf(f_texto, "%s\n", txt);
38      fprintf(f_texto, "%d\n", x);
39      fprintf(f_texto, "%d\n", y);
40      fprintf(f_texto, "%f\n", g);
41      fclose(f_texto);
42
43      // Los escribimos como datos binarios en Datos.dat
44      fwrite(txt, sizeof(txt), 1, f_data);
45      fwrite(&x, sizeof(x), 1, f_data);
46      fwrite(&y, sizeof(y), 1, f_data);
47      fwrite(&g, sizeof(g), 1, f_data);
```

```
48      fclose(f_data);
49
50      printf("Ficheros Datos.txt y Datos.dat creados\n");
51      return 0;
52    }
53    else { // Si hay argumento, modo "leer ficheros"
54      f_texto=fopen("Datos.txt", "r");
55      if (f_texto==NULL) {
56        perror("Al abrir Datos.txt");  exit(-1);
57      }
58      f_data=fopen("Datos.dat", "rb");
59      if (f_texto==NULL) {
60        perror("Al abrir Datos.dat"); exit(-1);
61      }
62      // Leer datos ASCII de Datos.txt
63      fscanf(f_texto, "%s\n", txt);
64      fscanf(f_texto, "%d\n", &x);
65      fscanf(f_texto, "%d\n", &y);
66      fscanf(f_texto, "%f\n", &g);
67      fclose(f_texto);
68
69      printf("Datos leidos de Datos.txt\n");
70      printf(" txt=%s\n", txt);
71      printf(" x=%d\n", x);
```

```
72      printf(" y=%d\n", y);
73      printf(" g=%f\n", g);
74
75      // Leemos los datos binarios de Datos.dat
76      fread(txt, 5, 1, f_data);
77      fread(&x, sizeof(int), 1, f_data);
78      fread(&y, sizeof(int), 1, f_data);
79      fread(&g, sizeof(float), 1, f_data);
80      fclose(f_data);
81
82      printf("Datos leidos de Datos.dat\n");
83      printf(" txt=%s\n", txt);
84      printf(" x=%d\n", x);
85      printf(" y=%d\n", y);
86      printf(" g=%f\n", g);
87      return 0;
88    }
89  }
```

Tras ejecutar el programa anterior encontraremos los ficheros `Datos.txt` y `Datos.dat`, sobre los que se pueden efectuar los siguientes experimentos:

- Volcar en pantalla el contenido de `Datos.txt`, usando el comando `cat`. Se comprueba que contiene texto legible.
- Hacer lo mismo con `Datos.dat`. ¿Sabrías explicar lo que ha salido?
- Examinar los bytes que hay almacenados en `Datos.txt` con ayuda de la herramienta `od`

  ```
  $ od -t "cxC" Datos.txt
  ```

  Se comprueba que tiene almacenados códigos ASCII, y el código `0a` que representa el salto de línea.
- Hacer lo mismo con `Datos.dat`. Comprobar que contiene codificado en ASCII la cadena, pero en binario los demás datos. Fijarse en el orden ¿*little* o *big endian*?

```
1  // Ejemplo de acceso a ficheros usando las funciones de bajo nivel
2  // (open/read/write)
3  //
4  // La funcion es la misma que la del programa ficheros.c, de hecho,
5  // los ficheros creados con uno de ellos deberian ser legibles por el
6  // otro.
7  //
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <sys/types.h>
11 #include <sys/stat.h>
12 #include <fcntl.h>
13 #include <unistd.h> // Para open/read/write en unix
14 #include <string.h>
15 int main(int argc, char *argv[])
16 {
17   int f_texto; // Ahora el tipo fichero es "int" !
18   int f_data;
19   int x=10;
20   int y=47;
21   float g=1.0;
22   char txt[5]="Hola"; // 5 = 4 letras mas terminador
23   char linea[80];     // Para convertir los datos a ASCII
```

```
24
25    if (argc<2) { // No hay argumento, modo "crear ficheros"
26      // Observar que en Unix, ademas de un modo de apertura se debe
27      // indicar los permisos de acceso con que se creara el fichero
28      // S_WRITE indica permiso de escritura para el propietario (el
29      // creador)
30      f_texto=open("Datos.txt", O_WRONLY|O_CREAT, S_IWRITE);
31      if (f_texto==-1) {
32        perror("Creando Datos.txt"); exit(-1);
33      }
34      // Al crear el fichero para datos binarios, bajo Windows/DOS,
35      // seria necesario añadir |O_BINARY al modo de apertura.
36      f_data=open("Datos.dat", O_WRONLY|O_CREAT, S_IWRITE);
37      if (f_data==-1) {
38        perror("Creando Datos.dat"); exit(-1);
39      }
40
41      // Escribimos los datos como ASCII en Datos.txt
42      // sprintf lo convierte en cadena de texto, y write lo escribe en
43      // el fichero. Observar el uso de strlen en lugar de sizeof.
44      sprintf(linea, "%s\n", txt); write(f_texto, linea, strlen(linea));
45      sprintf(linea, "%d\n", x);   write(f_texto, linea, strlen(linea));
46      sprintf(linea, "%d\n", y);   write(f_texto, linea, strlen(linea));
47      sprintf(linea, "%f\n", g);   write(f_texto, linea, strlen(linea));
```

```
48      close(f_texto);
49
50      // Ahora escribimos los mismos datos directamente en binario
51      write(f_data, txt, 5); // La cadena tiene 4 letras mas terminador
52      write(f_data, &x, sizeof(int));
53      write(f_data, &y, sizeof(int));
54      write(f_data, &g, sizeof(float));
55      close(f_data);
56
57      printf("Ficheros Datos.txt y Datos.dat creados\n");
58      return 0;
59    }
60    else { // Si hay argumento, modo "leer ficheros"
61      char buffer[200]; // Para leer el fichero .txt
62      // Nota: segun esta implementacion, el fichero no puede tener mas
63      // de 200 bytes
64      int n_bytes_leidos;
65
66      f_texto=open("Datos.txt", O_RDONLY);
67      if (f_texto==-1) {
68        perror("Al abrir Datos.txt");  exit(-1);
69      }
70      f_data=open("Datos.dat", O_RDONLY);
71      if (f_texto==-1) {
```

```
72        perror("Al abrir Datos.dat"); exit(-1);
73      }
74
75      // Leemos los datos ASCII de Datos.txt
76      // Leemos todo el fichero en un buffer, y extraemos la informacion
77      // de ese buffer usando sscanf.
78      n_bytes_leidos=read(f_texto, buffer, 200);
79      printf("Se han leido %d bytes del fichero Datos.txt\n",
80             n_bytes_leidos);
81      close(f_texto);
82      // Extraer datos del buffer, usando los tipos apropiados
83      sscanf(buffer, "%s\n%d\n%d\n%f\n", txt, &x, &y, &g);
84
85      printf("Datos leidos de Datos.txt\n");
86      printf(" txt=%s\n", txt);
87      printf(" x=%d\n", x);
88      printf(" y=%d\n", y);
89      printf(" g=%f\n", g);
90
91      // Leer los datos binarios de Datos.dat es mas sencillo
92      read(f_data, txt, 5);
93      read(f_data, &x, sizeof(int));
94      read(f_data, &y, sizeof(int));
95      read(f_data, &g, sizeof(float));
```

```
96      close(f_data);
97
98      printf("Datos leidos de Datos.dat\n");
99      printf(" txt=%s\n", txt);
100     printf(" x=%d\n", x);
101     printf(" y=%d\n", y);
102     printf(" g=%f\n", g);
103     return 0;
104   }
105 }
```

```c
// Experimentos con cadenas (strings) en C
#include <stdio.h>
#include <string.h>
#include <malloc.h>

int main()
{
  // Una cadena se puede guardar en un array de caracteres, o en una
  // zona de memoria a la que se hace apuntar un puntero a caracteres.

  char diez[10]="Holas"; // Reservamos 10 caracteres, aunque usamos
                         // solo 6 (letras mas terminador)
  char fija[]="Holas";   // El compliador calcula aqui cuantos bytes
                         // reservar (seran 6)
  char *puntero="Holas"; // Aqui la cadena se almacena en el area de
                         // "strings estaticos", y se hace que el
                         // puntero apunte a ella

  // Sorpresa?
  printf("Tamaño de diez=%d\n", sizeof(diez));
  printf("Tamaño de fija=%d\n", sizeof(fija));
  printf("Tamaño de puntero=%d\n", sizeof(puntero)); // !Eepa!

```

```
24    // Aqui no hay sorpresa
25    printf("Cadena diez.    Longitud=%d, contenido=|%s|\n",
26           strlen(diez), diez);
27    printf("Cadena fija.    Longitud=%d, contenido=|%s|\n",
28           strlen(fija), fija);
29    printf("Cadena puntero. Longitud=%d, contenido=|%s|\n",
30           strlen(puntero), puntero);
31
32    // La sintaxis para accedera a una letra cualquiera es la misma en
33    // los tres casos.
34    printf("Tercera letra: diez=%c, fija=%c, puntero=%c\n",
35           diez[2], fija[2], puntero[2]);
36
37    // La zona de memoria donde se almacenan los arrays es la misma (la
38    // pila de programa durante la ejecucion de main). En cambio la
39    // cadena estatica va a otra zona (la zona de strings estaticos)
40    printf("Direcciones de memoria donde se almacena el texto:\n");
41    printf("  diez   =%p\n", diez);
42    printf("  fija   =%p\n", fija);
43    printf("  puntero=%p\n", puntero);
44    printf(" &puntero=%p\n", &puntero); // Observar que el puntero se
45    // almacena en la pila, al ser una variable de main, aunque la
46    // direccion a que apunta no esta en la pila.
47
```

```
48    // Modificaciones de las cadenas.
49    printf("Añadir texto a la cadena fija [error grave]\n");
50    strcat(fija, ", caracolas");
51    printf("Resultado:\n");  // Inesperado?
52    printf(" Cadena fija.    Longitud=%d, contenido=|%s|\n",
53           strlen(fija), fija);
54    printf(" Cadena diez.    Longitud=%d, contenido=|%s|\n",
55           strlen(diez), diez);
56
57    printf("Añadir texto a la cadena puntero [mal hecho]\n");
58    strcat(puntero, ", caracolas");
59    printf("Resultado:\n");
60    printf(" Cadena puntero. Longitud=%d, contenido=|%s|\n",
61           strlen(puntero), puntero);
62    // Lo anterior puede dar un core o no, depende de la maquina
63
64    printf("Añadir texto a la cadena puntero [bien hecho]\n");
65    puntero=(char *) malloc(strlen(puntero)+strlen(", caracolas")+1);
66    strcpy(puntero, "Holas");
67    strcat(puntero, ", caracolas");
68    printf("Resultado:\n");
69    printf(" Cadena puntero. Longitud=%d, contenido=|%s|\n",
70           strlen(puntero), puntero);
71    // Aqui no hay problema. Es la forma correcta de hacerlo. Pero ahora
```

```
72    // el puntero ya no apunta a la zona de strings estaticos, sino al
73    // "heap" (la zona de memoria sobre la que opera malloc/free). La
74    // cadena original "Holas", sigue intacta en la zona de stings
75    // estaticos. El puntero a ella se ha perdido para siempre.
76
77    return 0;
78  }
```

- Este ejemplo implementa dos funciones y un programa principal que las llama.
- El codigo fuente de cada funcion va a un fichero diferente (y un tercero tiene el codigo de `main()`).
- Los ficheros de cabecera (`.h`) contienen los prototipos de las funciones
- Cada fuente (`.c`) se compila (sin enlazar) y genera un codigo objeto (`.o`).
- Los codigos objeto se enlazan en un solo ejecutable. O bien se crea una biblioteca (`.a`) con las los codigos gobjeto de las funciones, y se enlaza el principal con la biblioteca.
- Estas tareas se pueden automatizar con un `Makefile`.

```c
// Ejemplo de compilación separada. Programa principal.
#include <stdio.h>
// Los includes siguientes son necesarios para que el compilador sepa
// los prototipos de las funciones, y por tanto las conversiones de
// tipo que debe hacer (si fueran necesarias).
// ¡Prueba a quitar estas líneas!
#include "multiplica.h"
#include "divide.h"

int main()
{
  double x, y;
  double resultado;

  printf("Introduce un par de números reales:");
  scanf("%lf %lf", &x, &y);  // %lf es para doubles
  resultado=multiplica(x,y);
  printf("Su producto es %lf\n", resultado);
  resultado=divide(x,y);
  printf("Su cociente es %lf\n", resultado);
  return 0;
}
```

```
1  // Fichero de cabecera que proporciona el prototipo de la función
2  // multiplica()
3  #ifndef _MULTIPLICA_H_
4  #define _MULTIPLICA_H_
5
6  double multiplica(double, double);
7  #endif
```

```
1  // Fichero de cabecera que proporciona el prototipo de la función
2  // divide()
3  #ifndef _DIVIDE_H_
4  #define _DIVIDE_H_
5
6  double divide(double, double);
7  #endif
```

Observar el truco del `#ifndef` para prevenir que este fichero se incluya dos veces o más.

```c
// Implementación de la función multiplica()
double multiplica(double a, double b)
{
  return a*b;
}
```

```c
// Implementación de la función divide()
double divide(double a, double b)
{
  return a/b;
}
```

- Compilar sólo, sin enlazar (opción `-c`)

```
$ gcc -c programa.c
```

- Enlazar varios objetos en un solo ejecutable

```
$ gcc -o ejecutable obj1.o obj2.o ...
```

- Crear una biblioteca (comando `ar`)

```
$ ar r libbiblioteca.a obj1.o obj2.o ...
```

- Enlazar con una biblioteca (opción `-l`)

```
$ gcc -o ejecutable -Lruta objs.o ... -lbiblioteca
```

  Observar que el nombre de la biblioteca siempre debe empezar por `lib` y ese prefijo no se escribe en la opción `-l`

- Otra opción interesante: `-Wall` (máximo nivel de *warnings*)

```
1  # Opciones por defecto para el compilador. Solo se usan cuando es make
2  # quien deduce que el compilador debe ser llamado
3  CFLAGS=-Wall
4
5  # Compilador de C por defecto
6  CC=gcc
7
8  # Objetivos por defecto
9  all: calcular
10
11 # Este objetivo depende tambien de una libreria, asi que debemos
12 # indicar como se construye
13 calcular: principal.o libcalculos.a
14         $(CC) -o calcular principal.o  -lcalculos -L.
15
16 # Como se crea la librería
17 libcalculos.a: multiplica.o divide.o
18         ar r libcalculos.a multiplica.o divide.o
19
20 # Cómo se crean los .o no es necesario indicarlo. make tiene ese
21 # conocimiento. Los compilará con el compilador $(CC) usando las
22 # opciones $(CFLAGS)
23
```

```
24 # Reglas para hacer limpieza. Borrar ficheros auxiliares
25 clean:
26         -rm *.o *~ *.a Datos*
27
28 # O borrar todo y dejar solo los fuentes
29 cleanall: clean
30         -rm parametros error ficheros readwrite calcular cadenas
```