Tema 1

Gestión básica de ficheros

1.1. Introducción

Hasta ahora, los programas que se han hecho en la asignatura tomaban los datos, generalmente, de la denominada entrada *standard*, el teclado y devolvían el resultado escribiendo en la salida *standard*, la pantalla del monitor. Este mecanismo de comunicación de los programas, de entrada/salida de los datos tiene un inconveniente fundamental: ni la entrada ni la salida son reutilizables. Por un lado, cada vez que se ejecuta el programa hay que volver a teclear los datos y, por otro, tampoco se pueden conservar los datos de salida mostrados en pantalla.

La solución a este problema es usar un mecanismo que permita almacenar de forma permanente los datos, tanto de entrada como de salida. Este mecanismo van a ser los ficheros. Ya se ha hecho un primer uso de los ficheros. Los programas de temas anteriores se han guardado en un fichero para no tener que teclearlos cada vez que se van a ejecutar.

1.2. Flujos de entrada/salida

Este tema se centra en el uso de los ficheros como flujos de datos de un programa, tanto de entrada, para proporcionar los datos necesarios para realizar la tarea que ejecuta el programa, como de salida, para almacenar los datos obtenidos por el programa.

Se puede considerar un flujo como una secuencia de caracteres. El flujo de entrada de datos a los programas hasta ahora ha sido a través de `cin` y el flujo de salida a través de `cout`. `cin` representa el flujo de entrada de caracteres que viene del teclado y `cout` representa el flujo de salida a la pantalla del monitor.

En este tema se presenta cómo definir nuevos flujos para que la entrada/salida del programa se haga usando ficheros.

1.3. E/S de ficheros

1.3.1. Declaración de flujos

La biblioteca en la que se definen los tipos y funciones a usar para los flujos con ficheros es fstream:

```
#include <fstream>
```

Los flujos `cin` y `cout` ya están definidos en `#include <iostream>` por lo que el usuario no tiene que definirlo explícitamente en su programa.

Los flujos de entrada son el tipo `ifstream` y los de salida `ofstream`. Se definen como cualquier otra variable:

```
ifstream flujo_ent;
ofstream flujo_sal;
```

1.3.2. Apertura y cierre de flujos

Un flujo tiene que estar asociado a un fichero. El fichero se identifica mediante un nombre, que consta de una cadena de caracteres. La función para asociar el flujo y el fichero es `open`.

```
flujo_ent.open("archivo_datos.dat");
flujo_sal.open("archivo_salida.sal");[1]
```

La llamada a `open` no sigue la sintaxis habitual de las funciones como se han visto hasta ahora, sino que usan un punto entre el nombre de la variable y el nombre de la función. Esto se debe a que los flujos son de un tipo especial de datos, llamados objetos, que se estudiarán en un tema posterior.

Si el flujo se abre para escritura y el fichero ya existe, se pierde su contenido y se trabaja como si el fichero estuviera vacío.

Una vez que se ha abierto el flujo, está asociado al fichero cuyo nombre se pasa como parámetro. El nombre de fichero no vuelve a usar para acceder al fichero, sino que todos los accesos se hacen a través del flujo. Si el flujo es de entrada, se pueden ejecutar operaciones para leer datos desde ese fichero. Si es de salida, se puede escribir en el fichero a través de llamadas al funciones sobre el flujo.

Cuando se ha terminado de trabajar con un fichero, se ha de terminar la asociación entre el flujo y el fichero para dejar el flujo libre. Para ello se usa la operación `close`.

```
flujo_ent.close();
flujo_sal.close();
```

Un flujo puede volver a asociarse a otro fichero tras haber cerrado la conexión con el anterior.

Otros modos de apertura de ficheros

El uso anterior de la función `open` es el uso por defecto. Se puede modificar su comportamiento especificando un segundo parámetro en el que se indica el modo en el que se abre el fichero.

```
void open(const char p[], openmode modo);
```

Un fichero se puede abrir en modo:

- entrada. Con el modo `ios::in`. Permite hacer operaciones de lectura.
- salida. Con el modo `ios::out`. Permite hacer operaciones de escritura. Borra el contenido previo del fichero.
- añadir. Con el modo `ios::app`. Permite hacer operaciones de escritura. Añade la nueva información tras el contenido actual del fichero.

Como segundo parámetro se puede especificar una combinación de varias de las anteriores opciones.

```
fstream finout; finout.open(nombre_fichero,ios::in|ios::out);
```

Esta llamada a `open` tiene varios elementos que merecen un comentario más detallado.

En primer lugar, se ha llamado a `open` con dos parámetros, en vez de uno como se vio en la sección 1.3.2. Lo que ocurre en realidad es que el segundo parámetro de la función tiene un valor por defecto, es decir, si se llama con un solo parámetro, el segundo toma un valor predefinido. En este caso ese valor es que se abre en modo texto y que es de lectura, `ios::in`, para los flujos de tipo `ifstream` y de escritura, `ios::out`, para los de tipo `ofstream`.

Otra característica novedosa es la forma de los campos pasados en el segundo parámetro `ios::in`. El operador `::` indica el ámbito en el que se declara un recurso, ya sea una variable, una función o, como en este caso, una constante. Esto permite que se puedan usar en un mismo programa recursos con el mismo nombre definidos en, por ejemplo, diferentes bibliotecas. `ios::in` significa que la constante `in` se ha definido en el ámbito de `ios`. En C++ los ámbitos se llaman *namespaces*.

El tercer punto de interés es el operador `|`, que no se debe confundir con la disyunción lógica `||`. `|` es el operador *or* a nivel de *bits*. En el ejemplo sirve para decir que la apertura se hace con todas la opciones que se pasan como segundo parámetro.

Finalmente, hacer notar que la variable flujo es de tipo `fstream`, diferente a los dos vistos anteriormente. Este tipo de flujos permite tanto operaciones de lectura como de escritura.

1.3.3. Entrada/Salida básica

En temas anteriores se ha visto cómo leer y escribir datos con los operadores >> y << aplicados a los flujos `cin` y `cout`. Esas mismas operaciones están disponible para los flujos sobre ficheros de entrada y salida, respectivamente.

```
int i;
flujo_ent >> i;
flujo_sal << i;
```

En el ejemplo anterior, se lee un número entero del flujo de entrada y se escribe en el de salida.

Los operadores >> y << están definidos para los tipos predefinidos de C++: `char`, `int`, `short`, `long`, `unsigned int`, `unsigned short`, `unsigned long`, `unsigned float`, `unsigned double`, `unsigned long double` y `bool`. También están definidas para arrays de caracteres.

Cuando se abre un flujo de entrada, los datos empiezan a leerse desde el principio. Cuando se abre de salida, los datos se escriben tras los de la escritura previa.

El siguiente programa lee cinco números enteros de un fichero y los escribe en pantalla.

```
#include <iostream>
#include <fstream>
using namespace std; int main()
{
                        const int cant_nums = 5;
                        const char nombre_fichero[] = "datos.in";
                        int i, num;
                        ifstream f_ent;
                        f_ent.open(nombre_fichero);
                        for (i = 0; i <cant_nums; i++)
                        {
                            f_ent >> num;
                            cout << num;
                        }
                        f_ent.close();
}
```

1.3.4. Otras funciones sobre flujos

Hay otras funciones sobre flujos que tienen cometidos muy concretos:

- `fail`. Esta función se puede usar después de `open` para saber si la apertura se ha hecho correctamente. Si se intenta abrir para lectura un fichero que no existe fallará. Si se intenta abrir para escritura un fichero en un directorio en el que no tenemos permiso fallará.

  `fail` también informa si ha habido algún error al hacer una lectura, si, por ejemplo, hay una letra en un flujo de entrada cuando se pretende leer un entero.

- `eof`. Esta función nos dice si se ha llegado al final de fichero conectado a un flujo de entrada. Esta función devuelve `true` si no se ha podido hacer la lectura correctamente porque se ha llegado al final del flujo sin encontrar datos para leer. Si se ha podido hacer la lectura correctamente devuelve `false`.

Con estas funciones podemos hacer un programa que lea el contenido de un fichero de enteros separados por espacios y los vuelque en pantalla.

```
#include <iostream>
#include <fstream>
using namespace std; int main()
{
                        const char nombre_fichero[] = "datos.in";
                        int num;
                        ifstream f_ent;
                        f_ent.open(nombre_fichero);
                        if (f_ent.fail())
                        {
                           cout << "imposible abrir el fichero de entrada"<< endl;
                           return -1; // acabamos el programa por error irrecuperable
                        }
                        f_ent >> num;
                        while (!f_ent.eof())
                        {
                              cout << num;
                              f_ent >> num;
                        }
                        f_ent.close();
}
```

En el ejemplo se hace una lectura adelantada. Primero se lee el dato y después se comprueba mediante la función `eof` que ha sido correcta. Si lo ha sido, se escribe el dato en pantalla y se lee otro número entero.

1.3.5. Entrada/Salida de caracteres

El ejemplo anterior se puede modificar para hacer que muestre el contenido de un fichero en pantalla. Basta con cambiar la variable por una variable de tipo carácter (`char c;`) y hacer la lectura / escritura con ella.

Sin embargo, tenemos un problema. La operación >> aplicada a caracteres se salta los caracteres separadores, como los espacios en blanco, los tabuladores y los caracteres de nueva línea. Por tanto, la salida por pantalla serían todos los caracteres del fichero de entrada mostrados sin espacios intermedios y en una misma línea.

Para hacer la tarea que queremos hay que usar la función `get`, que lee un carácter del flujo de entrada, pero sin saltar los separadores. El programa para mostrar el contenido de un fichero en pantalla queda:

```
#include <iostream>
#include <fstream>
using namespace std; int main()
{
                        const char nombre_fichero[] = "datos.in";
                        char car;
                        ifstream f_ent;
                        f_ent.open(nombre_fichero);
                        if (f_ent.fail())
                        {
                           cout << "imposible abrir el fichero de entrada"<< end
                           return -1; // acabamos el programa por error irrecupera
                        }
                        f_ent.get(car);
                        while (!f_ent.eof())
                        {
                              cout << car;
                              f_ent.get(car);
                        }
                        f_ent.close();
}
```

La función equivalente para escribir un carácter en un flujo de salida es `put`, que, al

igual que `get`, toma un carácter como parámetro.

La operación `getline(char str[], int tam)` lee una cadena de caracteres como máximo hasta `tam` - 1 o hasta un final de linea. Añade el terminador char(0) al final de `str`. El carácter de final de línea se elimina del flujo.

1.3.6. Paso de parámetros

Cuando se define una función que toma un flujo tipo `ifstream` u `ofstream`, el parámetro se ha de pasar siempre por referencia. Por ejemplo:

```
#include <iostream>
#include <fstream>
using namespace std; void volcar_fichero(ifstream &ff)
{
                          char cc;
                          ff.get(cc);
                          while (!ff.eof())
                          {
                                cout << cc;
                                f.get(cc);
                          }
}
int main()
{
    const char nombre_fichero[] = "datos.in";
    char car;
    ifstream f_ent;
    f_ent.open(nombre_fichero);
    if (f_ent.fail())
    {
       cout << "imposible abrir el fichero de entrada"<< endl;
       return -1; // acabamos el programa por error irrecuperable
    }
    volcar_fichero(f_ent);
    f_ent.close();
}
```

1.3.7. Ficheros binarios

Las funciones anteriores hacen la entrada/salida en modo texto. Eso significa que los valores numéricos que se escriben se convierten a una cadena de caracteres y es esa cadena la que se vuelca al flujo. Cuando se lee el proceso es el inverso, se lee una cadena de caracteres y se transforma en un número.

La entrada/salida en modo binario no hace esa conversión a cadena de caracteres, sino lo que hace es una copia del contenido de una zona de memoria al flujo. Los caracteres se vuelcan de la misma forma en ambos formatos. En la lectura se actualiza una zona de memoria directamente con los datos leídos del flujo.

Este método es más rápido al no hacer la conversión y los ficheros resultantes suelen ser más pequeños. Por ejemplo, si se escribe el número 3.1415927 en modo texto, ocupará el espacio de 9 caracteres (generalmente un carácter ocupa un *byte*). Si se escribe en modo binario, ocupa el espacio del número en memoria (este tamaño, dado por `sizeof(float)`, suele ser 4 bytes.

Para usar estas operaciones, los flujos han de abrirse en modo binario, pasando indicando en el segundo parámetro de la función `open` si es de lectura escritura y que se abre en modo binario.

```
ifstream f_bin_ent;
ofstream f_bin_sal;
f_bin_ent.open("datos_ent.bin",ios::in|ios::binary);
f_bin_sal.open("datos_sal.bin",ios::out|ios::binary);
```

Las operaciones de lectura/escritura en modo binario son:

- `read(char *datos, streamsize tam)`. Esta operación lee del flujo a la dirección de memoria apuntada por el puntero `datos`. Lee tantos *bytes* como indique el parámetro `tam`.

- `write(const char *datos, streamsize tam)`. Esta operación escribe en el flujo desde la dirección de memoria apuntada por el puntero `datos`. Escribe tantos *bytes* como indique el parámetro `tam`.

La potencia de estas operaciones radica en que se puede pasar como primer parámetro la dirección de un valor de cualquier tipo, por lo que se puede escribir directamente un carácter, un número, o incluso un array o una estructura. Para indicar la dirección de una variable se usa el operador `&`. Hay que hacer notar que, como por motivos de eficiencia, cuando en C se pasa un array como parámetro, en realidad se pasa un puntero al primer elemento. Esto hace que no sea necesario usar `&` con los arrays.

La cantidad de memoria a escribir se calcula automáticamente con la operación `sizeof`, que se puede aplicar tanto a variables como a tipos.

El siguiente código lee números de teclado hasta leer un 0 y los guarda en un fichero en formato binario.

```
#include <iostream>
#include <fstream>
using namespace std; int main()
{
                        const char nombre_fichero[] = "datos.bin";
                        int num;
                        ofstream f_bin;
                        f_bin.open(nombre_fichero,ios::out|ios::binary);
                        if (f_bin.fail())
                        {
                           cout << "imposible abrir el fichero de salida"<< endl;
                           return -1; // acabamos el programa por error irrecuperable
                        }
                        cin >> num;
                        while (num != 0)
                        {
                              f_bin.write(&num, sizeof(int));
                              cin << num;
                        }
                        f_bin.close();
}
```

1.3.8. Acceso directo

Las lecturas y escrituras que hemos hecho hasta este momento se hacen de manera secuencial. Según se va leyendo, se va avanzando por el fichero y no se puede volver atrás a leer otra vez datos leídos anteriormente. Igualmente, cuando usamos un flujo de salida para escribir en un fichero, los datos se van escribiendo al final del fichero sin posibilidad de volver atrás para actualizar un dato escrito previamente. Para manejar los ficheros de manera no secuencial se usan las funciones de acceso directo.

Estas funciones nos permiten saber en qué posición de un flujo estamos leyendo o escribiendo y cambiar la posición en la que se va a producir a siguiente lectura o escritura.

- `pos_type tellg()`. Devuelve la posición del flujo en la que se va a realizar la siguiente lectura.
- `pos_type tellp()`. Devuelve la posición del flujo en la que se va a realizar la siguiente escritura.
- `seekg(pos_type pos)`. Hace que la siguiente lectura se haga en la posición `pos` del flujo.
- `seekg(off_type offset, ios_base::seekdir)`. Hace que la siguiente lectura se haga en una posición relativa a la indicada por el segundo parámetro, que puede ser `ios:beg`, inicio del flujo, `ios::cur`, la posición actual o `ios::end`, al final del fichero.
- `seekp(pos_type pos)`. Hace que la siguiente escritura se haga en la posición `pos` del flujo.
- `seekp(off_type offset, ios_base::seekdir)`. Hace que la siguiente escritura se haga en una posición relativa a la indicada por el segundo parámetro, que puede ser `ios:beg`, inicio del flujo, `ios::cur`, la posición actual o `ios::end`, al final del fichero.

Estas operaciones suelen usarse con ficheros en modo binario.

El siguiente ejemplo hace una búsqueda binaria de un número en un fichero que contiene una secuencia ordenada ascendentemente de números en modo binario.

```
#include <iostream>
#include <fstream>
```

```
using namespace std; int main()
{
                        const char nombre_fichero[] = "numeros.bin";
                        int num, dato;
                        unsigned long int ini_pos, fin_pos, medio;
                        ifstream f_bin;
                        f_bin.open(nombre_fichero,ios::in|ios::binary);
                        if (f_bin.fail())
                        {
                           cout << "imposible abrir el fichero de entrada"<< endl;
                           return -1; // acabamos el programa por error irrecuperable
                        }
                        ini_pos = 0;
                        // Para saber el tamaño del fichero me posiciono al final
                        f_bin.seekg(0, ios::end);
                        fin_pos = f_bin.tellg() / sizeof(int);
                        medio = (ini_pos + fin_pos) / 2;
                        cout << .Escriba el numero a buscar: ";
                        cin >> num;
                        cout << endl;
                        f_bin.seekg(medio * sizeof(int));
                        f_bin.read(&dato, sizeof(int));
                        while ((dato != num) && (ini_pos < fin_pos))
                        {
                              if (dato < num) // pasamos a la primera mitad
                              {
                                 ini_pos = medio + 1;
                              }
                              else
                              {
                                 fin_pos = medio - 1;
                              }
                              medio = (ini_pos + fin_pos) / 2;
                              f_bin.seekg(medio * sizeof(int));
                              f_bin.read(&dato, sizeof(int));
                        }
```

```
                    f_bin.close();
                    if (dato == num)
                    {
                       cout << "dato encontrado en la posicion: "<< ini_pos
                    }
                    else
                    {
                       cout << "dato NO encontrado"<< endl;
                    }
}
```

1.4. Manejos de ficheros con funciones C

Hay otra forma de manejar ficheros, que es la habitual del lenguaje C. Aunque la entrada y salida *standard* (generalmente el teclado y la pantalla del monitor, respectivamente) tienen sus propias operaciones de entrada/salida, se pueden usar las operaciones que se van a describir a continuación usando como parámetro para el flujo las constantes `stdin` para la entrada y `stdout` para la salida.

1.4.1. Declaración de flujos

Los flujos son variables del tipo `FILE *`. Para asociar un flujo un fichero se usa la operación `fopen`:

```
FILE *fopen(const char nombrefichero[], const char modo[])
```

Esta función devuelve un flujo asociado al fichero `nombrefichero`. Si ha habido algún error devuelve el puntero NULL.

Los posibles modos de apertura son los siguientes:

"r". Abre el fichero sólo para lectura. Se empieza a leer desde el principio del fichero.

"w". Abre el fichero sólo para escritura. Se pierde el contenido previo del fichero.

"a". Abre el fichero sólo para agregar información. Empieza a escribir a partir del final del fichero.

A cualquiera de esos modos se le puede añadir la letra '+' para indicar que se puede leer y escribir en el fichero.

Si se quiere abrir el fichero en modo binario, se añade la letra 'b' al final de la cadena de modo.

Tras acabar el procesamiento del fichero se cierra el flujo con la operación `fclose`:

```
int fclose(FILE *f)
```

1.4.2. Entrada/Salida básica

La entrada/salida básica se hace mediante la lectura/escritura de caracteres o arrays de caracteres. Las principales funciones son las siguientes:

- `int fgetc(FILE *f)`. Esta función devuelve el siguiente carácter del flujo o `EOF` si no hay más caracteres que leer en el flujo.[2]
- `char *fgets(char s[], int n, FILE *f)`. Esta función lee caracteres del flujo `f` al array `s`. Lee hasta encontrar un retorno de carro o leer `n`-1 caracteres.
- `int fputc(int c, FILE *f)`. Esta función escribe el carácter `c` en el flujo `f`.[3]
- `char *fputs(const char s[], FILE *f)`. Esta función escribe la cadena `s` en el flujo `f`.

1.4.3. Salida con formato

Las funciones anteriores sólo escriben caracteres, o cadenas de caracteres y, además, sin formato. La operación `fprintf` permite escribir valores numéricos y caracteres y darle un formato para hacer más legible la salida.

```
int fprintf(FILE *f, const char formato[], ...)
```

[2] `EOF` es una constante de tipo `int`.

[3] Aunque el parámetro formal es de tipo `int`, el parámetro real es de tipo `char`. Ambos tipos son compatibles en C.

`fprintf` tiene un número variable de parámetros. El primer parámetro es el flujo donde se van a escribir los datos. El segundo es una cadena de caracteres donde se indica el formato con el que se van a escribir.

El formato se compone de caracteres corrientes, que son volcados al flujo y de códigos de conversión. Simplificando, los códigos de conversión se componen del caracter `%` y de un segundo carácter que indica el tipo del dato que se va a escribir en el flujo.

`%c` para valores de tipo `char`

`%d` para valores de tipo `int`

`%f` para valores de tipo `float`

`%s` para cadenas de caracteres.

La cadena de formato admite caracteres especiales como el tabulador (indicado por `\t` o nueva línea `\n`.

Por cada código de conversión debe haber un parámetro en la llamada, del tipo especificado. Por ejemplo:

```
int num_linea;
char nom_fich[TAMNOMFICH];
fprintf(stdout, .ERROR de compilacion en la linea%d del fichero%s.\n",
num_linea, nom_fich);
```

puede producir una salida como:

```
ERROR de compilacion en la linea 28 del fichero columnas.cpp.
```

La operación para lectura con formato es `fscanf`:

```
int fscanf(FILE *f, const char formato[], ...)
```

`fscanf` es análoga a `fprintf`. También tiene un número variable de parámetros. El primer parámetro es el flujo de donde se van a leer los datos y el segundo es una cadena de caracteres donde se indica el formato con el que se van a leer.

La principal diferencia está en que los parámetros que siguen al formato tienen que indicar la dirección de memoria de una variable del tipo especificado en la cadena de

formato. Esto se suele hacer usando el operador `&` para indicar la dirección de memoria de la variable, como se vio en el tema anterior.

Para leer un entero, un flotante y una cadena de caracteres de un flujo se escribirá:

```
int n;
float x;
char cad[TAMCAD];
fscanf(f, "%d%f%s", &i, &x, cad)
```

Un error muy usual a la hora de usar `fscanf` es pasar directamente las variables en vez de su dirección, lo que provoca un comportamiento impredecible:

`fscanf(f, "%d%f", i, x) //ERRÓNEO`

1.4.4. Lectura/Escritura en modo binario

Las operaciones son:

- `size_t fread(void *ptr, size_t tam, size_t nobj, FILE *f)`. Esta operación lee del flujo a la zona de memoria apuntada por `ptr` hasta `nobjs` de tamaño `tam`. Devuelve el número de objetos que ha podido leer.
- `size_t fwrite(const void *ptr, size_t tam, size_t nobj, FILE *f)`. Esta operación escribe al flujo desde la posición de memoria apuntada por `ptr` hasta `nobjs` de tamaño `tam`. Devuelve el número de objetos que ha podido escribir.

La potencia de estas funciones es que el primer parámetro, el valor escrito o leído, puede ser de cualquier tipo, tanto un tipo básico como un `array` o un `struct`.

Para escribir un número real y un array de enteros se escribirá:

```
float d;
int ai[TAMARR];
fwrite((void *)&d, sizeof(float), 1, f);
fwrite((void *)ai, sizeof(int), TAMARR, f);
```

Como hay que pasar un puntero, al escribir el número real se usa el operador `&` para indicar la dirección de la variable. En el caso del `array` ya se ha visto que no es necesario.

1.4.5. Acceso directo

Las operaciones para conocer y colocar la posición de lectura/escritura del flujo son:

- `long ftell(FILE *f)`. Devuelve la posición actual de lectura/escritura.

- `int fseek(FILE *f, long offset, int origin)`. Actualiza la posición de lectura/escritura. La posición se fija a `offset` posiciones de `origin`, que debe ser `SEEK_SET` para el principio del fichero, `SEEK_CUR` para la posición actual o `SEEK_END` para el final del fichero.

1.5. Ejercicios

1. Escribir un programa que cuente el número de letras minúsculas, de letras mayúsculas y de cifras de un fichero. Nota: en la biblioteca `cctype` hay funciones que se pueden usar para este ejercicio.

2. Escribir un programa que copie el contenido de un fichero en otro.

3. Escribir una función que copie una línea de un flujo en otro. La función tendrá como parámetros el flujo en entrada, el flujo de salida y un `unsigned int` que indica cuál es el número de la línea que se va a copiar.

4. Escribir un programa con la opción de encriptar y de desencriptar un fichero de texto. La encriptación (codificación) consiste en que dado un fichero de texto de entrada genere otro fichero de salida de extensión <nombre>.COD donde el texto estará codificado (encriptado). Esta codificación consiste reemplazar cada carácter por el tercero siguiente (ej. 'a' → 'd'). Si el carácter resultante es no imprimible éste no se cambia. La opción de desencriptado consiste en leer un fichero <nombre>.COD y recuperar la información original.

5. Escribir un programa para procesar información sobre los clientes de una agencia matrimonial. El programa debe crear un fichero de texto llamado "PERSONAS.DAT" en el que se guarde la información de un número indeterminado de personas. La información que se guardará por cada persona será:

Nombre:	De 1 a 30 caracteres.
Edad	`unsigned int`.
Sexo	`char` (M/F).
Arte	`char` (S/N).
Deporte	`char` (S/N).
Libros	`char` (S/N).
Musica	`char` (S/N).

La información correspondiente a cada persona se leerá del teclado. El proceso finalizará cuando se teclee un campo `Nombre` que esté vacío.

6. Ampliar el programa que procesa clientes de una agencia matrimonial para que tome los datos de todos los candidatos a estudiar del fichero `PERSONAS.DAT` del ejercicio anterior, lea el cliente del teclado y finalmente genere como resultado un listado por pantalla con los nombres de los candidatos aceptados y un fichero llamado `ACEPTADOS.DAT` con toda la información correspondiente a los candidatos aceptados. Una persona del fichero `PERSONAS.DAT` se considerará aceptable como candidato si tiene diferente sexo y por lo menos tres aficiones comunes con respecto al aspirante introducido por pantalla. (El programa debe ser capaz de trabajar con cualquier número de personas en el fichero `PERSONAS.DAT`).

7. Codifique un programa que cree un fichero para contener los datos relativos a los artículos de un almacén. Para cada artículo habrá de guardar la siguiente información:

Código del artículo	(Numérico)
Nombre del artículo	(Cadena de caracteres)
Existencias actuales	(Numérico)
Precio	(Numérico)

Se deberán pedir datos de cada artículo por teclado hasta que como código se teclee 0. El fichero se habrá de crear ordenado por el código del articulo.

8. Escriba un programa que dados dos ficheros generados por el programa anterior y ordenados genere un tercer fichero que sea el resultado de mezclar de formar ordenada los dos primeros.

9. Escriba un programa que tome como entrada el fichero del ejercicio 4 y una condición sobre los campos `Existencias actuales` o `Precio`. La condición podrá ser: <campo> [<,<=,>,>=, <>] <número>

Este programa debe generar como salida un fichero llamado `salida.dat` que contenga todos aquellos artículos para los que se cumple la condición de entrada.

10. Escriba un programa que tenga como entrada un fichero que contenga un texto y dé como resultado una estadística de las palabras que hay de cada longitud, así como de los caracteres especiales (”,”, ”.”, ”:”, ”;”).