Sesión 1

E/S y punteros en lenguaje C

Objetivos

Manejo básico de las funciones de E/S estándar del lenguaje C .

Entender cómo son las variables de tipo puntero en el lenguaje C para Windows y para la arquitectura IA-32.

Comprender el uso del operador *indirección* (&).

Saber acceder al contenido de una variable mediante un puntero.

Procesar *arrays* mediante punteros.

Comprender cómo afectan a un puntero sus incrementos y decrementos, en función del tipo de dato al que apuntan.

Utilizar los punteros en el paso de argumentos a funciones.

1 Conocimientos previos

Antes de realizar esta práctica es necesario:

- Leer y comprender los apuntes proporcionados en las clases de teoría sobre las funciones *printf()* y *scanf_s()*.
- Leer y comprender los apuntes proporcionados en las clases de teoría sobre el uso de punteros en lenguaje C.

Desarrollo de la práctica

2 E/S estándar en C

2.1 La consola Win32

Los programas de tipo CUI hacen su salida en una consola Win32. Comenzaremos escribiendo el programa más sencillo posible en C (de tipo CUI) que lleve a cabo una operación de salida y observaremos la consola Win32 utilizada por el programa para llevara a cabo dicha salida.

H Crea un proyecto para un programa C, según se vio en el Apéndice B del bloque de prácticas anterior. Dale el nombre `2-1prog1`. Agrégale el fichero `2-1prog1.c`. Escribe en este fichero el siguiente código:

```
#include <stdio.h>

main()
{
    printf("Hola mundo");
}
```

Este es el programa más simple que podemos escribir que realiza una salida en consola. En este ejemplo, la función *printf()* utiliza una cadena de control sin parámetros de sustitución, por tanto, dicha cadena tal y como está es lo que va a ser enviado a la consola.

H Compila y enlaza el programa. Ahora vamos a ejecutar este programa sin depurarlo. Para ello existen dos posibilidades, o bien lo ejecutamos desde la interfaz de comandos textual del sistema operativo, o bien lo ejecutamos desde el explorador. Vamos a probar cada caso.

H Para ejecutarlo desde la interfaz de comandos textual, abre el programa CMD.EXE. Debes percatarte de que CMD.EXE es de tipo CUI y, por tanto, tiene una consola Win32 asociada, que es la ventana en la que se realiza el proceso de comunicación con el usuario. Ubica CMD.EXE en la carpeta *Debug* del proyecto. Ahora ejecuta el programa `2-1prog1.exe`. Observarás en la consola la cadena "Hola mundo". Fíjate que al ejecutarse este programa no se abre una nueva consola Win32, sino que la salida se hace en la consola Win32 de CMD.EXE. Lo que ocurre es que `2-1prog1.exe` ha heredado la consola de CMD.EXE y, por tanto, no necesita crear una nueva. El concepto de herencia de recursos entre programas se verá más adelante en la asignatura.

H Ahora vamos a ejecutar el programa desde el explorador de Windows. Para ello abre la carpeta *Debug* del proyecto, y para ejecutar el programa, haz doble clic sobre `2-1prog1.exe`. Observarás que se abre una consola Win32 y se cierra. Lo que ha ocurrido es que `2-1prog1.exe` se ha ejecutado, ha creado una consola Win32, ha

impreso la cadena “Hola mundo” en ella y ha terminado. Todo esto ocurre tan rápido que casi no da tiempo ni a observarlo.

Vamos a hacer una pequeña modificación en el programa para que antes de que termine de ejecutarse espere por una pulsación de teclado y así podamos observar la consola Win32 que genera durante su ejecución.

H En el programa `2-1prog1.c`, justo después de la llamada a la función *printf()*, escribe una llamada a la función *_getch()*. El objetivo de esta función, que no recibe parámetros, es esperar a que el usuario pulse una tecla, entonces la función retorna el código de la letra pulsada. No es necesario que recojas el valor devuelto por la función en ninguna variable, ya que lo único que esperamos de ella es que detenga la ejecución del programa. Esta función requiere que incluyas el fichero de cabecera `conio.h`. Tras hacer estas modificaciones en el programa, compílalo y enlázalo para obtener el ejecutable. Entonces ejecútalo, primero desde CMD.EXE. Observa cómo el programa se detiene hasta que pulses una tecla. Pulsa para que termine. Ahora ejecútalo desde el explorador de Windows, haciendo doble clic sobre su icono. Ahora sí debes observar la consola Win32 que genera el programa al ejecutarse. Esta consola es propia del programa y no heredada de CMD.EXE. Ahora cuando pulses una tecla el programa terminará y, por tanto, se cerrará su consola Win32 asociada.

2.2 Salida formateada: la función *printf()*

printf() es una función que trabaja con un número de parámetros variable. No obstante, como mínimo, debe recibir un parámetro, que es la cadena de control. Esta cadena puede estar formada por caracteres y parámetros de sustitución. En el caso del programa anterior, la cadena de control recibida por *printf()*, “Hola mundo”, estaba formada exclusivamente por caracteres, por tanto, lo que se enviaba a la pantalla era esa misma cadena sin modificación alguna. Ahora vamos a probar a introducir parámetros de sustitución dentro de la cadena de control, con objeto de que la función *printf()* nos permita sacar por pantalla el contenido de una variable. Además probaremos a utilizar especificadores de formato diferentes, para observar en la pantalla el contenido de la variable expresado en formas diversas. Para esto utilizaremos el siguiente esqueleto de programa.

```
#include <stdio.h>

main()
{
    char car = 'A';

      // Imprimir car interpretada como carácter
      printf(“\ncar interpretada como carácter = %c\n”, car);

      // Imprimir car interpretada como entero decimal


      // Imprimir car interpretada en hexadecimal


      // Imprimir la dirección de car

}
```

El programa que hiciste en la sección anterior, `2-1prog1.c`, tiene tan poca información que no merece la pena mantenerlo, así que utilizaremos ese mismo programa para incluir este listado.

H Borra el contenido de `2-1prog1.c` y, después, escribe en él el listado anterior. Ahora hay que completar el listado teniendo en cuenta los comentarios. Después de cada comentario debes escribir una sentencia *printf()*. A continuación se indica la salida que debe ser realizada por el programa:

```
car interpretada como caracter = A
car interpretada como entero decimal = 65
car interpretada en hexadecimal = 41
Direccion de car = 0012FFXX
```

Las dos últimos dígitos de la dirección (XX) pueden variar según dónde quede ubicada la variable. Para este ejemplo, necesitarás utilizar los especificadores de formato `%c`, `%d`, `%x`, `%p`. Utilizarás uno en cada *printf()*. La secuencia de escape '\n' te permitirá cambiar de línea.

Cuando hayas completado el programa, compílalo, enlázalo y ejecútalo desde CMD.EXE comprobando su correcto funcionamiento.

H Cambia el valor de inicialización de la variable `car`, utilizando 'B' en vez de 'A'. Compila, enlaza y vuelve a ejecutar al programa desde CMD.EXE, comprobando que los resultados obtenidos son coherentes.

2.3 Entrada formateada: la función *scanf_s()*

La función *scanf_s()* puede resultar compleja de utilizar, pero en estas prácticas, vamos usarla de la forma más simple posible. Utilizaremos *scanf_s()* para leer por teclado un valor y cargarlo en una variable, o bien para leer una cadena de caracteres y almacenarlo en un *array* de caracteres.

El esqueleto de programa que vamos a utilizar para probar la función *scanf_s()* se indica a continuación:

```
main()
{
    int A=0, B=0;
    char cadena[10];

    // PARTE PRIEMRA: carga de variables enteras

    // Cargar A interpretando la cadena en pantalla como decimal
    printf("Introduce A: ");
    scanf_s("%d", &A);

    // Cargar B interpretando la cadena en pantalla como hexadecimal



    // Imprimir A y B, ambas como decimal
```

```
    // PARTE SEGUNDA: carga de una cadena de caracteres
    // Cargar cadena con una cadena introducida por pantalla

    // Imprimir la cadena introducida
}
```

Este programa, según se indica en sus comentarios, tiene dos partes. El objetivo de la primera parte es probar la carga de variables de tipo entero, y el de la segunda parte, la carga de cadenas de caracteres.

En la primera parte de este programa tenemos dos variables `A` y `B`, cuyos valores queremos introducir por pantalla. En el listado se indica cómo cargar la variable A. Fíjate que previamente al *s_scanf()* utilizamos un *printf()* para enviar un mensaje a la consola que le indica al usuario lo que debe hacer. Fíjate también que *s_scanf()* no recibe como parámetro la variable que queremos cargar (A), sino su dirección (&A).

H Crea un proyecto para un programa C con el nombre `2-1prog2`. Agrégale el fichero `2-1prog2.c`. Escribe en este fichero el programa incompleto anterior. Escribe las sentencias necesarias para cargar la variable B siguiendo la indicación realizada en el comentario correspondiente. Escribe también la sentencia *printf()* que imprime A y B. Este sentencia debe imprimir la siguiente línea en la consola:

```
A = XX;  B = YY
```

Donde XX e YY representan los valores que hayan tomado A y B respectivamente. De momento no te preocupes de la segunda parte del programa, se rellenará después. Compila y enlaza el programa y ejecútalo desde CMD.EXE. Carga las dos variables con el mismo valor, por ejemplo 27, al imprimirlas en la consola, el programa debería proporcionarte:

```
A = 27;  B = 39
```

Si obtienes el resultado anterior, has hecho las cosas bien. No obstante, ¿comprendes el resultado obtenido? Si no es así, pregúntale a tu profesor. Puedes hacer alguna otra ejecución del programa, introduciendo otros valores y comprobando los resultados obtenidos.

H Ahora vamos a resolver la segunda parte del programa. Empieza comentando las sentencias de la primera parte, así nos centraremos nada más que en la segunda. Una vez comentadas estas sentencias, completa la segunda parte del programa siguiendo los comentarios. Recuerda que para cadenas de caracteres, el especificador de formato a utilizar es `%s`, y que cuando se carga una cadena mediante *scanf_s()* hay que indicar el tamaño del *array* en el que se carga. Compila y enlaza el programa y ejecútalo con CMD.EXE. Introduce la cadena HOLA. Observa que se carga en el *array* de caracteres, ya que luego se imprime correctamente. Haz otra prueba introduciendo la cadena HOLAHOLAHOLA. Si has programado correctamente la función *scanf_s()*, esta cadena no debe cargarse en el *array* `cadena`, ya que su longitud es mayor que el número de elementos del *array*. Comprueba que la cadena

máxima que puedes introducir debe tener 9 caracteres, ya que estos se completarán con el terminador de cadena dando un total de 10, que es el tamaño del *array*.

3 Punteros

3.1 Manejo básico de punteros

Empezaremos utilizando un puntero para acceder a una variable simple.

H Crea el proyecto `2-1prog3` y agrégale el fichero `2-1prog3.c`. Escribe en este fichero el programa que se indica a continuación:

```
#include <stdio.h>

main()
{
      int A=27;
      int *p;

      // Inicializacion de p con la direccion de A
      p=&A;

      // Ver contenido de A
      printf("Contenido de A (visto mediente A) = %d \n", A);
      printf("Contenido de A (visto mediante p) = %d \n", *p);
}
```

Observa que una vez que `p` ha sido inicializado con la dirección de `A`, podemos acceder al contenido de `A` mediante `p`. Fíjate que para acceder a donde el puntero apunta (o sea, a `A`, en nuestro ejemplo) hay que preceder el identificador del puntero con el operador '`*`'.

H Compila y enlaza `2-1prog3.c` y ejecútalo desde CMD.EXE, comprobando que su comportamiento es el esperado.

Ahora vamos a utilizar el puntero para modificar el contenido de `A`.

H En el programa anterior, introduce una sentencia que cargue la variable `A` con el valor 133, pero no debes utilizar `A` en la sentencia sino `p`. Esta nueva sentencia debes colocarla justo a continuación de la sentencia "`p=&A`". Una vez realizada esta modificación, compila y enlaza el programa de nuevo y ejecútalo desde CMD.EXE. Entonces comprueba que los resultados mostrados en su ejecución son consistentes.

3.2 Punteros y *arrays*

Según irás aprendiendo, uno de los objetivos básicos de los punteros es proporcionar una forma alternativa de acceso a estructuras de datos complejas, como por ejemplo los *arrays*. Para analizar cómo podemos procesar *arrays* utilizando punteros, vamos a partir de un ejemplo muy simple que se muestra a continuación:

```
#include <stdio.h>

char vocales[5]={'a', 'e', 'i', 'o', 'u'};
```

```
main()
{
      int i; // Contador

      for(i=0; i<5; i++)
      {
            printf("vocales[%d] = %c \n", i, vocales[i]);
      }
}
```

El objetivo de este programa es visualizar por pantalla las letras almacenadas en el *array* `vocales`. De momento no se utiliza ningún puntero en este programa.

H Asegúrate de que comprendes bien la sintaxis utilizada en la sentencia *printf()* del programa. Si tienes alguna duda pregúntale a tu profesor. Crea el proyecto `2-1prog4`, agrégale el fichero `2-1prog4.c` y escribe en este fichero el programa anterior. Entonces compílalo y enlázalo, ejecútalo desde CMD.EXE, y observa su funcionamiento.

Ahora vamos a hacer algunas modificaciones en el programa para acceder a los elementos del *array* mediante un puntero.

H Crea el proyecto `2-1prog5`, agrégale el fichero `2-1prog5.c` y copia en él el programa anterior. A continuación de la definición de la variable `i`, define un puntero, llamado `p`, para acceder a los elementos del *array*. Fíjate que al ser los elementos del *array* de tipo `char`, el puntero deberá ser también de tipo `char`. Comenta todo el bucle `for`: volveremos a utilizarlo más tarde, pero no ahora. Inicializa `p` para que apunte al elemento [0] del *array* (recuerda que el identificador del *array* representa su dirección de comienzo). Escribe una sentencia *printf()* que visualice la primera letra del *array*, pero para acceder a dicha letra sólo puedes utilizar el puntero `p`. El programa debe escribir en la consola

```
Vocales[0]=a
```

Compila y enlaza el programa, ejecútalo desde CMD.EXE y comprueba que funciona correctamente.

Vamos a acceder ahora con el puntero a otros elementos del *array*. Por ejemplo, imagina que queremos acceder al elemento [2] del *array*. Para esto tenemos dos posibilidades:

1) Modificar el puntero para que apunte al elemento deseado y acceder a dicho elemento mediante `*p`.
2) Utilizar el puntero y un desplazamiento apropiado. En ese caso habría que utilizar la expresión `*(p+2)`. Esto significa acceder a donde apunta `p` más un desplazamiento de dos unidades.

Vamos a probar estas dos posibilidades.

H En el programa anterior, añade, justo a continuación de la sentencia *printf()* que visualiza el elemento [0] del *array*, otra sentencia *printf()* que visualice el elemento. Utiliza para ello la sintaxis `*(p+2)`. Una vez realizada la modificación, compila y enlaza el programa, ejecútalo desde CMD.EXE y comprueba su correcto funcionamiento.

H ¿Serías capaz de predecir, qué salida generaría el segundo *printf()* del programa anterior si quitases los paréntesis de la expresión `*(p+2)`. Anota a continuación tu predicción:

Realiza dicha modificación en el programa, compílalo y enlázalo, ejecútalo desde CMD.EXE y comprueba que tu predicción es la correcta.

Toma buena nota de la importancia de los paréntesis en las expresiones que utilizan punteros.

Ahora vamos a acceder al elemento [2] del *array* incrementando el puntero.

H En este momento tienes en el programa `2-1prog5.c` dos sentencias *printf()* que imprimen los elementos [0] y [2] del *array*. Ahora vas a introducir entre esas dos sentencias otra sentencia que incremente `p` en 2. De este modo, `p` queda apuntando al elemento [2] del *array*. Modifica la segunda sentencia *printf()* de la forma apropiada para que visualice el elemento [2] del *array*. Compila y enlaza el programa, ejecútalo desde CMD.EXE y comprueba que su comportamiento es el esperado.

Ahora ya debes estar en condiciones de volver a escribir el programa `2-1prog4.c`, pero procesando los elementos del *array* mediante un puntero.

H Crea el proyecto `2-1prog6` y agrégale el fichero `2-1prog6.c`. Copia en este fichero el código del programa `2-1prog4.c`. Ahora tienes que hacer las modificaciones necesarias en el programa para procesar el *array* `vocales` mediante un puntero. Esto significa que en la sentencia *printf()* que imprime las letras del *array* no puedes utilizar `vocales[i]` sino el puntero que definas para llevar a cabo el procesamiento. Una vez realizadas las modificaciones, compila y enlaza el programa, ejecútalo desde CMD.EXE y comprueba su correcto funcionamiento.

Ahora vamos a comprender un poco mejor la aritmética de punteros, es decir, que es lo que ocurre exactamente cuando incrementamos o decrementamos un puntero, o bien cuando le sumamos o restamos una cantidad entera. Para ello vamos a utilizar un programa en el que se definen *arrays* de diferentes tipos de datos. El listado de este programa se muestra a continuación.

```
#include <stdio.h>

char   vocales[5]={'a', 'e', 'i', 'o', 'u'};
int    enteros[5]={1, 2, 3, 4, 5};
double reales[5]={1.0, 2.0, 3.0, 4.0, 5.0};

main()
{
        // Definicion de un puntero para acceder a cada tipo de array
        char *p_vocales;
        int  *p_enteros;
        double *p_reales;

        // Inicializacion de los punteros
        p_vocales = vocales;
        p_enteros = enteros;
        p_reales = reales;
```

```
        // Acceso mediante p_vocales al array de vocales
        printf("Direccion de p_vocales = %p\n", p_vocales);
        printf("vocales[0] = %c\n", *p_vocales);
        p_vocales++;
        printf("Direccion de p_vocales = %p\n", p_vocales);
        printf("vocales[1] = %c\n", *p_vocales);
}
```

Vamos a comentar algunas cosas interesantes de este programa. En primer lugar se definen en él tres *arrays* de tipos de datos diferentes: caracteres (`char`), enteros (`int`) y reales de doble precisión (`double`). Cada uno de estos tipos de datos necesita un número diferente de bytes para su almacenamiento en memoria: 1 byte para los datos de tipo carácter, 4 para los de tipo entero, y 8 para los reales de doble precisión.

Al reflexionar sobre el acceso a datos de diferentes tamaños se nos plantea la siguiente pregunta. Cuando accedemos mediante un puntero a una *array* de datos de un determinado tipo, ¿debemos tener en cuenta el tamaño de los datos a la hora llevar a cabo los incrementos y decrementos del puntero? Vamos a llevar a cabo unos experimentos acerca de esto para luego obtener una conclusión.

H Crea el proyecto `2-1prog7` y agrégale el fichero `2-1prog7.c`. Copia el listado del programa anterior en este fichero. De momento en este programa sólo se trabaja sobre el *array* `vocales`. En el programa se usa el puntero `p_vocales` para acceder a los dos primeros elementos del *array*. El programa muestra también las direcciones que toma el puntero cuando accede al elemento [0] y [1] del *array*. Compila y enlaza el programa, ejecútalo desde CMD.EXE y toma nota de los siguientes datos:

Tipo de dato al que se accede =

Valor de `p_vocales` en el acceso a `vocales[0]` =

Valor de `p_vocales` en el acceso a `vocales[1]` =

Valor del incremento del puntero =

Habrás observado que el puntero se incrementa en 1, lo cual es totalmente lógico porque apunta a datos que ocupan un byte, y que por tanto se almacenan en una sola posición de memoria. Vamos a ver ahora que ocurre con los otros *arrays*.

H En la parte final del programa anterior tienes el siguiente bloque de sentencias:

```
        // Acceso mediante p_vocales al array de vocales
        printf("Direccion de p_vocales = %p\n", p_vocales);
        printf("vocales[0] = %c\n", *p_vocales);
        p_vocales++;
        printf("Direccion de p_vocales = %p\n", p_vocales);
        printf("vocales[1] = %c\n", *p_vocales);
```

Haz una copia de todo este bloque de sentencias justo a continuación de él mismo. Comenta todas las sentencias del bloque original. Así quedan documentadas en el programa. Ahora en el segundo bloque de sentencias vas a realizar las modificaciones necesarias para acceder al *array* de enteros en lugar de al *array* de vocales. Primero cambia el comentario por este otro: *// Acceso mediante p_enteros al array de enteros*. Ahora tienes que modificar las sentencias *printf()* para sacar por consola, mediante el puntero `p_enteros`, los elementos [0] y [1] del *array* de enteros. Puede asaltarte la siguiente duda: ¿en cuánto hay que incrementar `p_enteros`? Como ahora los datos del *array* son de 4 bytes cada uno, ¿habrá que

incrementar el puntero en 4? La respuesta es no: hay que seguir incrementando el puntero en 1. Vamos a ver experimentalmente porque esto es así y luego extraeremos las conclusiones pertinentes. Haz las modificaciones pedidas en el programa, compílalo, ejecútalo y toma nota de los siguientes datos:

Tipo de dato al que se accede =

Valor de `p_enteros` en el acceso a `enteros[0]` =

Valor de `p_enteros` en el acceso a `enteros[1]` =

Valor del incremento del puntero =

Habrás observado que aunque en el programa incrementas el puntero `p_enteros` en 1, en realidad resulta incrementado en 4. Antes de extraer una conclusión vamos a hacer el último experimento acerca de este comportamiento de los punteros. Vamos ahora a acceder al *array* de reales.

H Para ello, haz una copia de todo el bloque de sentencias que sacan por consola los elementos del *array* `p_enteros`, justo a continuación de sí mismo. Comenta todas las sentencias del bloque original para que queden documentadas en el programa. Ahora en el bloque de sentencias que acabas de copiar, vas a realizar las modificaciones necesarias para acceder al *array* de reales en vez de al *array* de enteros. Primero cambia el comentario por este otro: *// Acceso mediante p_reales al array de reales*. Ahora tienes que modificar las sentencias *printf()* para sacar por consola, mediante el puntero `p_reales`, los elementos [0] y [1] del *array* de reales. Ten en cuenta que para imprimir datos de tipo `double`, se requiere el parámetro de sustitución `%lf` en la cadena de control de *printf()*. Al igual que en el caso anterior debes incrementar `p_reales` sólo en 1. Haz las modificaciones pedidas en el programa, compílalo, ejecútalo y toma nota de los siguientes datos:

Tipo de dato al que se accede =

Valor de `p_reales` en el acceso a `reales[0]` =

Valor de `p_reales` en el acceso a `reales[1]` =

Valor del incremento del puntero =

A tenor de los experimentos realizados, la conclusión es la siguiente:

Un puntero se define para apuntar a datos de un determinado tipo. Por tanto, cuando se incrementa un puntero en una unidad, se incrementa la dirección que contiene en el valor necesario para apuntar al siguiente dato del tipo especificado. Exactamente lo mismo se aplica para los decrementos.

3.3 Paso de argumentos a funciones

Hasta ahora hemos practicado el funcionamiento básico de los punteros, pero no hemos visto nada aún que no se pueda hacer sin punteros. Ahora es el momento de ver la utilidad de los punteros, y te darás cuenta que sin ellos el lenguaje C sería de escasa utilidad.

Imagínate que quieres programar una función genérica que procese una cadena de caracteres. El procesamiento a realizar es convertir la cadena a mayúsculas. La función

debe procesar en principio cualquier cadena, la cual debe estar finalizada con un terminador que será el número 0. ¿Qué necesita la función para procesar la cadena? Parece que lo más simple es proporcionarle a la función la dirección de memoria en la que se encuentra la cadena y programar la función para realizar el procesamiento basándose en dicha dirección. Si no existiesen los punteros no habría forma de programar esta función en lenguaje C.

Vamos a ver cómo se podría programar esta función. Para ello, se muestra a continuación el listado de un programa formado por dos funciones: *convierte()* que tiene por objetivo convertir una cadena a mayúsculas, y *main()* que llama a *convierte()* para procesar una cadena concreta.

```
#include <stdio.h>

// Prototipo
void convierte( char * );

main()
{
        char cadena[6]={'a', 'e', 'D', 'r', 'Z', 0};

        // Imprimir cadena antes de procesarla
        printf("Cadena = %s\n", cadena);

        // Llamar a convierte
        ...

        printf("Cadena = %s\n", cadena);
}

void convierte( char *p_cadena )
{
        int i=0;

        while( ... )
        {
            ...
            ...
        }
}
```

Como puedes observar en el listado anterior, la función *convierte()* recibe como parámetro un puntero a carácter. Este parámetro tiene un doble cometido:

1) Indicarle a la función la dirección en la que se encuentra la estructura de datos a procesar.
2) Indicarle a la función que los elementos integrantes de la estructura de datos son de tipo `char`.

No hay otra forma posible de programar la función *convierte()* que no sea mediante la utilización de un puntero a `char`.

H Crea el proyecto `2-1prog8` y agrégale el fichero `2-1prog8.c`. Copia el listado del programa anterior en este fichero. Ahora debes completar el programa. Para ello ten en cuenta las siguientes indicaciones:

- Para convertir los caracteres de la cadena a mayúsculas, lo más simple es llevar a cabo una operación *and* con una máscara. En C la operación *and* se indica mediante el operador `&`.

- Para indicar la máscara será conveniente utilizar una constante hexadecimal. En C las constantes hexadecimales se escriben empezando por '0x'. Así por ejemplo para escribir 15 en hexadecimal se utiliza la constante 0xF. Observa que esta sintaxis es diferente a la utilizada en el lenguaje ensamblador.
- Para determinar la máscara puedes comparar el código ASCII de 'A', que es 0x41, con el de 'a', que es 0x61.

Completa el programa, compílalo y enlázalo, ejecútalo desde CMD.EXE y comprueba su correcto funcionamiento.

H Haz las modificaciones necesarias en el programa anterior para que procese dos cadenas en lugar de una. Las cadenas a procesar deben ser:

```
char cadena[6]={'a', 'e', 'D', 'r', 'Z', 0};
char cadena2[8]={'i', 'N', 'a', 'Q', 'q', 's', 'P', 0};
```

El programa debe sacar por consola las dos cadenas antes y después de ser procesadas.

Para terminar, hay que volver a resaltar que el objetivo fundamental de los punteros es proporcionar un mecanismo para pasar direcciones de estructuras de datos a funciones, con objeto de que las funciones puedan procesar dichas estructuras.

4 Ejercicios adicionales

E La función *convierte()* del programa anterior está pensada para procesar cadenas que contengan solamente letras mayúsculas y minúsculas. Sin embargo, si la cadena contiene otros caracteres, estos pueden resultar transformados por la función, lo cual no tendría sentido. Modifica la función *convierte()* para que procese correctamente cualquier cadena, sean cuales sean los caracteres que contenga. Prueba tu programa con varias cadenas diferentes. (Datos: ascii[a] = 0x61; ascii[z] = 0x7A).

E El objetivo de este ejercicio es desarrollar una función para copiar cadenas. El prototipo de esta función es el siguiente:

```
void copia_cadenas(char *destino, char *origen);
```

Haz un programa en el que implementes el código necesario para esta función. Utiliza la función *main()* del programa para procesar las siguientes cadenas:

```
char cad_or[8]={'1', 'A', 'B', '#', 'q', 'i', '+', 0};
char cad_des[80];
```

Usa *printf()* para imprimir la cadena origen y la cadena destino después de la llamada a *copia_cadenas()*, comprobando que ambas cadenas son iguales.