Universidad de Alcalá

Departamento de Automática

INFORMÁTICA

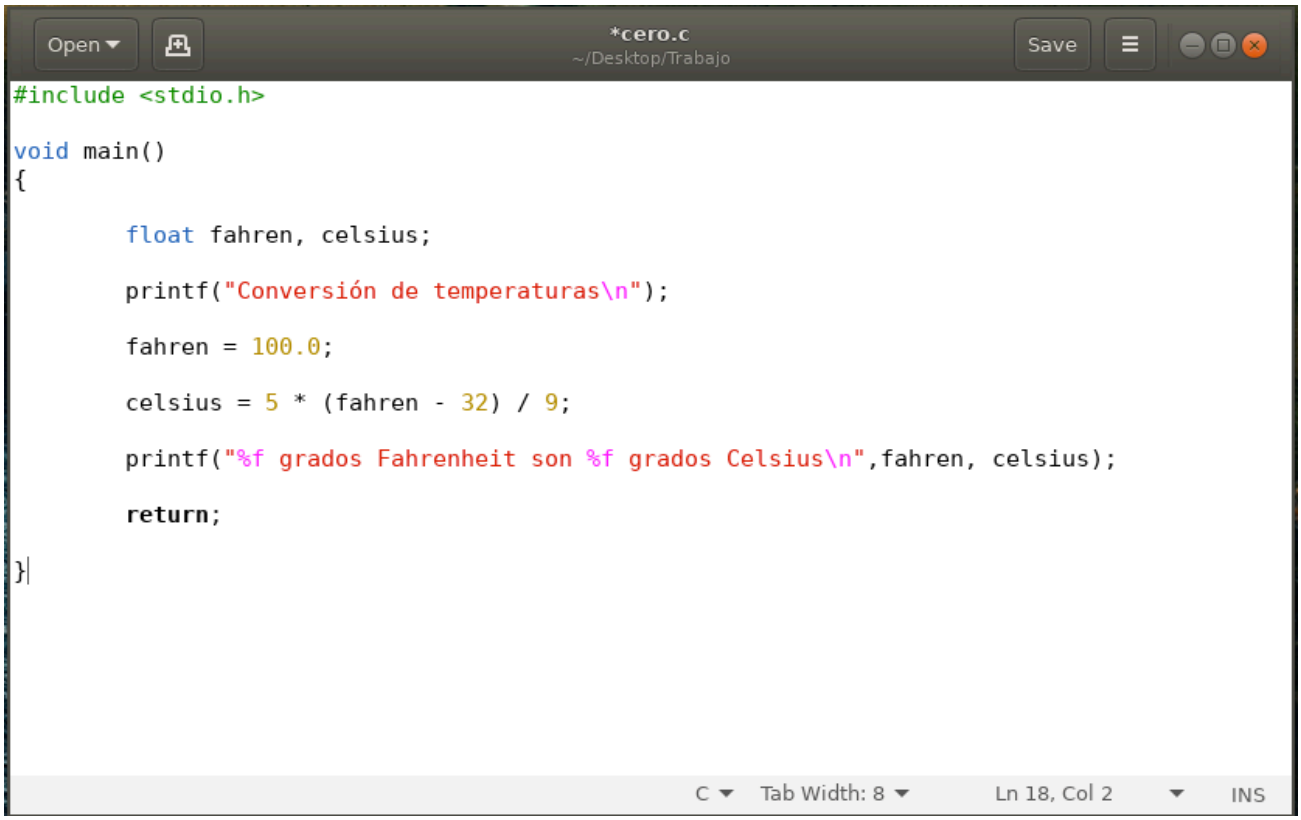
Práctica 2.
Desarrollo de programas.
Entrada y salida estándar en C.

Grado en Ingeniería en Electrónica y Automática Industrial
Curso 2023-2024
v2.7 (08.09.23)

CREACIÓN DE UN PROGRAMA

A la hora de realizar los ejercicios de las prácticas posteriores, la primera labor es el diseño del algoritmo. Una vez se tiene claro el algoritmo, se puede pasar a la creación del programa, que será una mera traslación a código del mismo. En el caso de los ejercicios más sencillos, el programa estará contenido en un único fichero fuente. Para crearlo se puede utilizar cualquier editor de texto sin formato, como `vim` o `emacs`. En el caso de este cuaderno, se va a utilizar un editor gráfico presente en la distribución de Ubuntu que se ha seleccionado llamado `gedit`.

Una vez arrancado el editor, se escribe el programa. Conviene ir salvando el trabajo de vez en cuando para evitar pérdidas si se va la luz o causas similares. El nombre del archivo fuente debe tener la extensión `.c`. En este ejemplo, se usa como nombre `muestra.c`. La siguiente figura muestra el editor de texto con el código del programa de ejemplo.



Una vez escrito el programa, debe compilarse y "*linkarse*" (del verbo inglés *to link*, enlazar, que se utiliza en la terminología inglesa). Se trata de dos operaciones diferentes, pero en el caso de un programa con un único fichero fuente se puede hacer con una única invocación al programa `gcc`. `gcc` es una aplicación que engloba (entre otras cosas) el compilador y el "*linkador*". La forma de invocarlo para este ejemplo es:

```
bash-ln.05$ gcc -g -o muestra muestra.c
```

La opción `-g` le indica que genere información de depuración (ya que, en caso contrario no se genera, y por lo tanto no será posible depurarlo después). La opción `-o nombre` le indica que debe poner el nombre `nombre` al fichero ejecutable resultado de la compilación. En este caso, `muestra`. El último argumento de la línea es el nombre del fichero fuente que hay que compilar.

Si el programa tiene errores de sintaxis, `gcc` indicará las líneas en la que se encuentran y aportará una pequeña descripción del problema. En caso de que no encuentre errores de sintaxis, el resultado será el archivo ejecutable.

GESTIÓN DE LA ENTRADA Y SALIDA ESTÁNDAR EN C.

En este apartado se va a estudiar cómo se utilizan la entrada y la salida estándar en un programa en C. El punto de partida es el programa básico que se ha editado en el apartado anterior, y, a partir de él, se exponen las cuestiones más relevantes de la entrada y salida.

Esquema básico de un programa en C.

La estructura básica de un programa sencillo en C es la del programa de la diapositiva 17 del capítulo 4, sección 1 que se introdujo en el editor en el apartado anterior. Se muestra a continuación. En el código mostrado se incluyen los números de línea, que no es necesario añadir a la hora de crear el programa en el editor.

```
1 /* Conversión de una temperatura en grados Fahrenheit a grados Celsius. */
2
3 #include <stdio.h>
4
5 int main()
6 {
7    int fahren, celsius;    /* Variables enteras */
8
9    printf("Conversión de °F a °C:\n");
10
11   /* Temperatura a convertir */
12   fahren = 100;
13
14   /* Conversión */
15   celsius = 5*(fahren-32)/9;
16
17   /* Mostrar resultados */
18   printf("%d °F = %d °C\n",fahren, celsius);
19   return 0;
20 }
```

En el programa se distinguen 2 secciones principales: la función `main` (la segunda sección, de la línea 5 en adelante), y lo que va antes de ella (la primera, líneas 1 a 4). En ésta última se incluyen las directivas de inclusión de bibliotecas (como `stdio.h` en el ejemplo) y las declaraciones de variables globales (que en el ejemplo no existen). En la función `main` aparecen las sentencias del programa principal.

En esta práctica se van a tratar algunas de las funciones de entrada/salida más importantes en C. Estas funciones trabajan con la entrada o la salida estándar. Por omisión, éstas son el teclado y la pantalla, pero pueden ser redirigidas desde la línea de comandos. Todas las funciones están en la librería `stdio.h`, por lo que, para usarlas, será necesario incluir la directiva de inclusión correspondiente en el programa.

En el programa propuesto, las líneas que comienzan por la palabra `printf` (9 y 18) están realizando operaciones de salida. La palabra `printf` responde al nombre de una función, que se encarga de volcar en la salida estándar lo que se le indique. La descripción completa de la función `printf`, y otras funciones para salida estándar se describen a continuación.

Salida estándar.

La principal función para salida estándar de datos es `printf`. La sintaxis es:

```
int printf(const char *formato, …)
```

La función permite volcar, sobre la salida estándar, los datos que se le indican, y formatear esa salida de acuerdo a las directrices que se le dan. Estas directrices se especifican en el primer argumento de la función. Se trata de una cadena de control, y está formada por dos tipos de elementos: caracteres ASCII y secuencias de control de formato. Los caracteres ASCII se vuelcan tal cual, y las secuencias de control de formato se sustituyen por el resto de argumentos de la función en el orden en que aparecen. Cada secuencia de control de formato comienza con el carácter '`%`', y a continuación incluye la información sobre cómo formatear la salida del dato que le corresponde. El formato de cada secuencia de control de formato es:

```
%[flags][ancho][.precisión][prefijo-tipo]formato
```

A continuación se explica cada uno de los campos:

`flags` (opcional)
- «`-`» justifica a la izquierda
- «`+`» fuerza la aparición del signo siempre
- «`0`» completa con ceros a la izquierda todo el campo

`ancho` (opcional): ancho del campo en el que aparecerá el dato

`precisión` (opcional)
- en enteros, número de dígitos
- en reales, número de dígitos decimales
- en cadenas, número de caracteres.

`prefijo-tipo` (opcional) indica a la función cómo debe interpretar el dato contenido en la memoria:
- «`h`» Interpreta un short
- «`l`» Interpreta long en los enteros o double en los reales
- «`L`» Interpreta un long double

`formato` (obligatorio) especifica el tipo de dato de la variable cuyo contenido se va a mostrar
- «`d`» entero con signo mostrado en decimal
- «`u`» entero sin signo mostrado en decimal
- «`o`» entero sin signo mostrado en octal

«x» entero sin signo mostrado en hexadecimal (X para mayúsculas)
«f» número real en formato [-]ddd.ddd
«e» número real en formato [-]d.ddde[±]ddd (E para mayúsculas)
«g» número real en el formato más corto
«c» carácter
«s» cadena de caracteres

De esta manera, la función printf invocada en la línea 9 no contiene secuencias de control de formato, con lo que simplemente escribirá en pantalla (en salida estándar, por ser más precisos) los caracteres que forman la cadena de control tal cual.

La llamada de la línea 18, por el contrario, sí tiene dos secuencias de control de formato. En concreto, aparece en dos ocasiones la secuencia '%d'. Se trata de dos secuencias sencillas en las que únicamente se especifica el formato, es decir, el tipo de dato de las variables que hay que visualizar. En este caso, se trata de dos enteros (d). Las variables que se muestran son las que figuran a continuación de la cadena de control, por orden: fahren (primera secuencia de control) y celsius (segunda secuencia). Los valores de ambas variables se "incrustan" en las posiciones de la cadena de control indicadas por las dos secuencias de control de formato '%d' (cada variable en su posición).

Las secuencias de control de formato permiten especificar, con un alto nivel de detalle, cómo se quieren mostrar los datos. A continuación se muestran varios ejemplos con secuencias de control de formato más complejas.

Ejemplos:
Si se tiene un entero, dato, para escribirlo en 8 posiciones, completando con ceros a la izquierda hasta 6 dígitos y en formato entero con signo:

```
printf("%8.6d",dato);
```

Para escribir dato, un entero sin signo, completando con ceros a la izquierda hasta 8 posiciones, justificado a la izquierda en un espacio de 12 posiciones:

```
printf("%-12.8u",dato);
```

Para escribir dato, un entero, en formato hexadecimal (en complemento a 2):

```
printf("%x",dato);
```

El siguiente ejemplo muestra por pantalla una cadena de caracteres creada como variable del programa. Una cadena de caracteres es una colección de caracteres dispuestos uno a continuación de otro, de forma que todos juntos muestran un mensaje.

```
char cadena[11]="Una cadena";

printf("%s",cadena);
```

En el ejemplo, la variable `cadena` se declara de tipo `char`, el correspondiente a caracteres. El número entre corchetes (11) es el máximo de caracteres que la cadena puede tener. Este número debe ser mayor en, al menos, una unidad que el número de caracteres que se quieren almacenar, en este caso 10, los que forman la colección "Una cadena". Con esta declaración se puede hacer la invocación a la función `printf` para indicar que escriba por pantalla los caracteres que se quieren mostrar simplemente utilizando el nombre de la colección, que en este caso es `cadena`.

Las secuencias de control de formato también se pueden incrustar en una cadena de caracteres ASCII:

```
unsigned int dato = -5;

printf("El dato es: %-12.8u\n",dato);
```

En este caso, los caracteres de la cadena se escriben en la salida estándar, y la variable `dato` se incrusta en el lugar indicado por el código de escape correspondiente ("`%-12.8u`").

`printf` es una función que permite un alto grado de control sobre la salida que se realiza. Para tareas más sencillas existen otras funciones más fáciles de usar.

`int puts(const char *)`: para escribir una simple cadena de texto.
`int putchar(int)`: para escribir un único carácter.

Entrada estándar.

Análoga a la función `printf` existe una función para entrada, `scanf`, con una funcionalidad relativamente extensa. El formato es:

```
int scanf(const char *formato, …)
```

En este caso, la cadena formato, que también es una cadena de control, no se vuelca a la salida, sino que corresponde a lo que se espera leer de la entrada. La función ignora los '*whitespace characters*' (caracteres de espacio, es decir: espacios, tabuladores y retornos de carro) que se encuentre antes del siguiente carácter. Un '*whitespace character*' que figure en la cadena de formato se asimila a cualquier cantidad de ellos en la entrada estándar (incluso ninguno). Igual que en `printf`, en `scanf` la cadena de control puede incluir secuencias de control de formato, en este caso con la sintaxis:

```
%[*][ancho][prefijo-tipo] formato
```

Los campos que lo forman son:

`*`: Indica que el dato se lee, pero no se asigna a ninguna variable.

`ancho` (opcional): Número de caracteres a leer (se ignoran los restantes)

`prefijo-tipo` (opcional): modifica el tipo de almacenamiento que se espera en función de la variable donde se va a almacenar el dato:

`h` unsigned short int

`hh` unsigned char
`l` unsigned long int
`ll` unsigned long long int
`L` long double

`formato` (obligatorio). Determina el tipo de dato. Los posibles valores son los mismos que en el campo análogo de `printf`().

Ejemplos:
Para leer un entero con signo de 6 dígitos:

```
scanf("%6d",&dato);
```

Para leer una cadena de caracteres:

```
char cadena[20];

scanf("%20s",cadena);
```

Para leer dos enteros:

```
scanf("%d %d",&dato1,&dato2);
```

En este último ejemplo, si se trabaja sobre teclado, el usuario deberá teclear primero el primer entero, a continuación un número arbitrario de '*whitespace characters*', y por último el segundo entero. Para hacérselo más intuitivo, es más conveniente leer primero el primer número y luego, con otra llamada a scanf, el segundo:

```
scanf("\n%d",&data1);
scanf("\n%d",&data2);
```

NOTA: la secuencia "`\n`" al principio de la cadena de formato sirve para eliminar los '*enters*' que se hayan pulsado previamente. Esto es útil cuando se deben leer varios datos seguidos (y marcando el final de cada uno con su correspondiente '*enter*').

De la misma forma que en la salida de datos, para la entrada existen funciones más sencillas:
`char *gets(char *)`: obtiene una cadena de texto.
`int getchar(void)`: obtiene un carácter.

EJERCICIOS.

1. Escribir un programa que haga lo siguiente: deberá tener una cadena de texto en una variable, con un valor inicial establecido por el programador, y mostrará esa variable por pantalla.
2. Escribir un programa que lea de teclado una cadena de texto (almacenándola en una variable) y muestre ese valor por pantalla. (Es posible crear una variable con una cadena de texto vacía simplemente omitiendo su valor inicial).

3. Escribir un programa que lea dos enteros con signo por teclado y muestre por pantalla su suma. Nota: es aconsejable leer los números uno por uno, cada uno con su llamada a la función de entrada, en lugar de ambos en la misma llamada.
4. Modificar el programa anterior para que trabaje con números sin signo. Tener en cuenta que el programa no puede evitar que el usuario introduzca números con signo. El usuario debe saber lo que hace cuando use el programa.
5. Modificar el programa anterior para que trabaje con números en coma flotante.
6. Escribir un programa que muestre por pantalla en hexadecimal un entero sin signo que lea por teclado.
7. Modificar el programa anterior para que lea por teclado un número entero con signo.