Laboratorio de Arquitectura de Redes

Entrada y salida estándar

Entrada y salida estándar

- Entradas y salidas
- Salida con formato: `printf()`
- Entrada de datos con formato: `scanf()`
  - El buffer de teclado
- Entrada de caracteres: `getchar()`
- Salida de caracteres: `putchar()`
- Entrada y salida de cadenas
  - Lectura de cadenas de caracteres
  - Escritura de cadenas de caracteres

Entradas y salidas (I)

- Cuando un programa está en ejecución, el procesador realiza operaciones de
  - Lectura en memoria de instrucciones
  - Lectura en memoria de datos
  - Procesado de información
  - Escritura en memoria de datos
  - Obtención de datos del exterior: *entradas*
  - Envío de datos al exterior: *salidas*

Entradas y salidas (II)

- Las *entradas* y *salidas* de información de un programa pueden realizarse:
  - Sobre las unidades de almacenamiento
    - Apertura del archivo
    - Operación de lectura/escritura sobre el archivo
    - Cierre del archivo
  - Sobre dispositivos periféricos
    - Directamente sobre los periféricos: operación hardware
    - A través de los controladores de los periféricos
    - Con los recursos ofrecidos por el sistema operativo

Entradas y salidas (III)

- Cuando un programa está en ejecución
  - Se convierte en un *proceso*
  - Tiene asociados por el sistema operativo:
    - Un *archivo estándar de entrada* asociado por defecto al teclado: **`stdin`**
      - Lo que se escribe en el archivo `stdin`, aparece directamente en la pantalla
    - Un *archivo estándar de salida* asociado por defecto a la pantalla: **`stdout`**
      - Lo que se lee del archivo `stdout`, se obtiene directamente del teclado
    - Un *archivo estándar para salida de errores* asociado por defecto a la pantalla: **`stderr`**
      - Los mensajes de error son enviados directamente a la pantalla, lugar físico del archivo `stderr`.

Entradas y salidas (IV)

- El estándar ANSI C creó un conjunto de funciones estándar para la entrada y salida a través de los ficheros estándar **`stdin`** y **`stdout`**
  - Están definidas en el archivo STDIO.H, por lo que para utilizarlas en un programa es preciso incluir la línea

    ```
    #include <stdio.h>
    ```

  - Las más importantes
    - `printf()` para escribir datos *con formato*
    - `scanf()` para leer datos con formato
    - `getchar()` para leer caracteres del teclado
    - `putchar()` para escribir caracteres en pantalla

Salida con formato: `printf()` (I)

- La función *devuelve* el número de bytes escritos o `EOF` en caso de error
- Plantilla de utilización:
  `printf("cadena de control", lista de argumentos)`
- Elementos en "`cadena de control`" :
  - Caracteres y símbolos ASCII normales
  - Caracteres *de escape* o **secuencias de barra invertida**
    - Comienzan siempre con símbolo «\»
  - **Especificadores de formato** para la representación de los valores almacenados en variables.
    - Comienzan siempre con el símbolo «%»
- La `lista de argumentos` es una relación de las variables (separadas por comas) cuyos contenidos se quiere mostrar
  - Debe haber el mismo número de variables que especificadores de formato

Salida con formato: printf() (II)

- Caracteres de escape o secuencias de barra invertida más habituales

\a	Alarma (pitido)	\'	Comilla simple
\b	Espacio atrás	\"	Comillas dobles
\f	Salto de página	\\	Barra invertida
\n	Nueva línea	\oo	Carácter ASCII en octal
\r	Retorno de línea	\xHH	Carácter ASCII en hexadecimal
\t	Tabulador	\0	Carácter nulo (Código ASCII cero)

Salida con formato: `printf()` (III)

- Sintaxis de los especificadores de formato (I)

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

  - `flags`. Opcional
    - «`-`» justifica a la izquierda
    - «`+`» fuerza la aparición del signo siempre
    - «`0`» Completa con ceros a la izquierda todo el campo
  - `anchura`. 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.

Salida con formato: printf() (IV)

- Sintaxis de los especificadores de formato (II)

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

  - `prefijo-tipo`. Para indicar a la función como 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`

Salida con formato: `printf()` (V)

- Sintaxis de los especificadores de formato (III)
  `%[flags][anchura][.precisión][prefijo-tipo] formato`
  - `formato`. Campo obligatorio, para determinar el tipo de dato de la variable cuyo contenido se va a mostrar
    - «d» Entero con signo en mostrado en decimal
    - «u» Entero sin signo mostrado en decimal
    - «o» Entero sin signo mostrado en octal
    - «x» Entero sin signo mostrado en hexadecimal
    - «f» Número real en formato `[–]ddd.ddd`
    - «e» Número real en formato `[–]d.dddE[±]ddd`
    - «g» Número real en el formato más corto
    - «c» Carácter
    - «s» Cadena de caracteres

Salida con formato: printf() (VI)

- Ejemplos

```
printf(“Número entero: %d”, edad);
printf(“Letra:%c \t Octal:%o \t Hexadecimal: %x”,
          código, código, código);
printf(“Número real: %f \t %E \t %G”,
          estatura, estatura, estatura);
```

Entrada con formato: scanf() (I)

- La función *devuelve* el número de datos leídos correctamente o EOF en caso de error
- Plantilla de utilización:

  `scanf("cadena de control", lista de argumentos)`

  - `"cadena de control"` :
    - No es una cadena que aparezca en pantalla
    - Se corresponde con la cadena que la función *espera* encontrarse en el teclado carácter a carácter
    - Incluye los caracteres de separación o *espacios en blanco q*ue separan los especificadores de formato. Pueden ser
      - El espacio en blanco « »
      - El tabulador «\t»
      - El retorno de carro «\n»
    - Incluye los **especificadores de formato** que provocan la captura de un dato

Entrada con formato: scanf() (II)

- Sintaxis de los especificadores de formato

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

  - `[*]`. Opcional: anula la asignación al siguiente dato

```
scanf("%d %*s", &valor); /* Lee el dato y la cadena
    que se teclee a continuación del valor entero,
    pero no se asigna a ninguna variable */
```

    - `anchura`. Opcional: Número de caracteres a leer (se ignoran los restantes
    - `prefijo-tipo`. Opcional:
      - En enteros, número de dígitos
      - En reales, número de dígitos decimales
      - En cadenas, número de caracteres
  - `formato`. Obligatorio. Determina el tipo de dato (igual que en `printf()`)

Entrada con formato: scanf() (III)

- Plantilla de utilización (II)

  `scanf("cadena de control", lista de argumentos)`

- La `lista de argumentos` es una relación de las **direcciones de memoria** de las variables (separadas por comas)
  - La dirección de una variable se obtiene precediendo el nombre de la variable del operador «&»
  - Debe haber el mismo número de variables que especificadores de formato

Entrada con formato: scanf() (IV)

- El **buffer de teclado** es una zona de almacenamiento intermedio asociada a la entrada estándar `stdin`
  - Se almacenan los códigos ASCII de las teclas pulsadas
  - Se valida cuando se pulsa la tecla INTRO
  - Sólo se eliminan del buffer los códigos de los caracteres leídos
  - Es la zona de memoria donde `scanf()` obtiene los valores
    - Puede que no se *cojan* todos los caracteres tecleados
    - Se quedan hasta la siguiente lectura del buffer del teclado
    - Hay una función definida en `STDIO.H` que borra este buffer

```
fflush(sdtin);
```

Entrada de caracteres: `getchar()` (I)

- Función para leer caracteres del teclado. Prototipo:

```
int getchar(void);
```

  - Definida en `STDIO.H`
  - No requiere ningún argumento
  - Devuelve el código ASCII de la tecla pulsada o `EOF` en caso de error
  - Solo finaliza, cogiendo un código, cuando se pulsa la tecla INTRO
    - Si se ha pulsado más de una tecla, quedan en el buffer del teclado

Entrada de caracteres: getchar() (II)

- Ejemplo:

```
char a;
a = getchar(void);
scanf("%c", &a);
```

  - Las dos sentencias son equivalentes
- Otras funciones similares
  - Definidas en CONIO.H
    - getch(). Lee directamente del teclado, sin que se muestre en pantalla y sin necesidad de pulsar INTRO
    - getche(). Como getch() pero sí se muestra el carácter en pantalla
- Las teclas especiales (F1 … F12) y las combinaciones con CTRL y ALT generan dos códigos de 8 bits, es decir, equivalen a dos pulsaciones (dos caracteres)

Salida de caracteres: putchar()

- Función para mostrar caracteres en la pantalla. Prototipo:

```
int putchar(variable);
```

  - Definida en STDIO.H
  - Como argumento requiere el nombre de la variable que contiene el código a mostrar
  - Devuelve en un entero el código ASCII del carácter mostrado o EOF en caso de error
- Ejemplo:

```
char a;
putchar(a);
printf("%c", a);
```

  - Las dos sentencias son equivalentes

Lectura y escritura de cadenas (I)

- En lenguaje C no existe un tipo de dato de *cadena de caracteres*
- Los *arrays de caracteres* o **cadenas de caracteres** son *vectores*
  - Cuyos elementos se almacenan consecutivamente en memoria
  - Se puede hacer referencia a todo el conjunto mediante un único identificador
  - Para referenciar a un elemento se utiliza el identificador y un índice entre corchetes

    `cadena[indice]`
  - El último elemento es siempre el carácter nulo «`\0`»
  - Declaración de un array (incluido el nulo final):

    `char nombrevariable[NUMERODEELEMENTOS];`
  - Se tratarán con detalle más adelante

Lectura y escritura de cadenas (II)

- Lectura de cadenas: `gets(cadena)`
  - Está definida en `STDIO.H`
  - Necesita, como argumento, el identificador de una cadena de caracteres
  - Lee todos los caracteres tecleados hasta el INTRO que lo recoge y lo sustituye por `'\0'` en memoria
  - Ejemplo

```
#define NUMELEM  100
char cadena[NUMELEM]; /* Declaración */
gets(cadena)   /* Lee una cadena */
scanf("%s", cadena);/* Lee una cadena */
```

    - Las dos sentencias son equivalentes
    - Leen todo lo tecleado (sea cual sea el valor de `NUMELEM`

Lectura y escritura de cadenas (III)

- La función `printf()` muestra en pantalla la cadena que va entre comillas como argumento
- Escritura de cadenas: `puts(cadena)`
  - Está definida en `STDIO.H`
  - Necesita, como argumento, el identificador de una cadena de caracteres o la propia cadena entre comillas dobles
  - Lleva a pantalla los símbolos ASCII de los códigos almacenados hasta encontrar el nulo `'\0'` que es sustituido por un INTRO
  - Ejemplo

```
puts(cadena)   /* Muestra una cadena */
printf("%s", cadena);/* Muestra una cadena */
```

    - Las dos sentencias son equivalentes