***Printf( ) man***

La función **printf** (que deriva su nombre de ***“print formatted”***) imprime un mensaje por pantalla utilizando una “cadena de formato” que incluye las instrucciones para mezclar múltiples cadenas en la cadena final a mostrar por pantalla. La función se define en la biblioteca estándar de C ***stdio.h*** y su prototipo es el siguiente:

**int** printf(const char \*format, ...);

printf es una ***función variádica*** ya que recibe un número variable de parámetros. El primer parámetro es fijo y es la cadena de formato. En ella se incluye texto a imprimir literalmente y ***marcador de posición*** a reemplazar por texto que se obtiene del segundo parámetro, que en este caso se representa con una elipsis ya que puede tener infinidad de variables.

Por tanto, printf se llama con tantos parámetros como marcadores de posición haya en la cadena de formato más la propia cadena de formato. El siguiente ejemplo muestra cómo se imprime el valor de la variable contador.

***printf*** *("El valor es %d.\n", contador)****;***

***Placeholder o marcador de posición:*** Son comandos especiales (%) que se introducen en la cadena de texto con un salto de línea(\n) seguido de una coma y el argumento que queramos imprimir o en su lugar la variable donde estará guardado su valor. Si en la cadena de formato aparecen varios marcadores de posición, los valores a incluir se toman en el mismo orden en el que aparecen.

Los ***marcadores de posición*** en la cadena de formato deben tener la siguiente ***estructura*** (los campos entre corchetes son optativos):

***%[****parameter****][****flags****][****width****][.****precision****][****length****]type***

Cada uno de los nombres (*parameter, flags, width, precision, length* y *type*) representa un conjunto de valores posibles que se explican a continuación:

|  |  |
| --- | --- |
| **Parameter** | **Descripción** |
| n$ | Se reemplaza “n” por un número para cambiar el orden en el que se procesan los argumentos. Por ejemplo %3$d se refiere al tercer argumento independientemente del lugar que ocupa en la cadena de formato. |
| **Flags** | ***Descripción*** |
| número | Rellena con espacios (o con ceros, ver siguiente flag) a la izquierda hasta el valor del número. |
| 0 | Se rellena con ceros a la izquierda hasta el valor dado por el flag anterior. Por ejemplo “%03d” imprime un número justificado con ceros hasta tres dígitos. |
| + | Imprimir el signo de un número |
| - | Justifica el campo a la izquierda (por defecto ya hemos dicho que se justifica a la derecha) |
| # | Formato alternativo. Para reales se dejan ceros al final y se imprime siempre la coma. Para números que no están en base 10, se añade un prefijo denotando la base. |

|  |  |
| --- | --- |
| **Width** | **Descripción** |
| número | Tamaño del ancho del campo donde se imprimirá el valor. |
| \* | Igual que el caso anterior, pero el número a utilizar se pasa como parámetro justo antes del valor. Por ejemplo printf("%\*d", 5, 10) imprime el número 10, pero con un ancho de cinco dígitos (es decir, rellenará con 3 espacios en blanco a la izquierda). |
| **Precision** | ***Descripción*** |
| número | Tamaño de la parte decimal para números reales. Número de caracteres a imprimir para cadenas de texto |
| \* | Igual que el caso anterior, pero el número a utilizar se pasa como parámetro justo antes del valor. Por ejemplo printf("%.\*s", 3, "abcdef") imprime “abc”. |
| **Length** | ***Descripción*** |
| hh | Convertir variable de tipo char a entero e imprimir |
| h | Convertir variable de tipo short a entero e imprimir |
| l | Para enteros, se espera una variable de tipo long. |
| ll | Para enteros, se espera una variable de tipo long long. |
| L | Para reales, se espera una variable de tipo long double. |
| z | Para enteros, se espera un argumento de tipo size\_t. |
| Type | Descripción |
| %c | Imprime el carácter ASCII correspondiente |
| %d, %i | Conversión decimal con signo de un entero |
| %x, %X | Conversión hexadecimal sin signo minúscula o mayúscula |
| %p | Dirección de memoria (puntero) |
| %e, %E | Conversión a coma flotante con signo en notación científica |
| %f, %F | Conversión a coma flotante con signo, usando punto decimal |
| %g, %G | Conversión a coma flotante, usando la notación que requiera menor espacio |
| %o | Conversión octal sin signo de un entero |
| %u | Conversión decimal sin signo de un entero |
| %s | Cadena de caracteres (terminada en '\0') |
| %% | Imprime el símbolo % |

***Ejemplos de marcas de formato***

Los marcadores de formato que se incluyen como parte de la cadena que se pasa como primer parámetro a printf ofrece muchas posibilidades. A continuación, se muestran algunos ejemplos:

***Especificar el ancho mínimo:*** Podemos poner un entero entre el símbolo de porcentaje (%) y el especificador de formato, para indicar que la salida alcance un ancho mínimo. Por ejemplo, **“%10f”** asegura que la salida va a tener al menos 10 espacios de ancho. Esto es útil cuando se van a imprimir datos en forma de columnas.

Por ejemplo, si tenemos:

int num = 12;

int num2 = 12345;

printf("%d\n”, num2);

printf("%5d\n", num);Se imprime:

12345

12

***Alinear la salida:*** Por defecto, la salida cuando se especifica ancho mínimo está justificada a la derecha. Para justificarla a la izquierda, hay que preceder el dígito de la anchura con el signo menos (-). Por ejemplo, “**%-12d”** especifica un ancho mínimo de 12, saca la salida justificada a la izquierda.

***Especificador de precisión:*** Puedes poner un punto (.) y un entero después de especificar un ancho de campo mínimo para especificar la precisión. En un dato de tipo float, esto permite especificar el número de decimales a sacar. En un dato de tipo entero o en cadenas de caracteres, especifica el ancho o longitud máxima.

***Listas enlazadas***

Las ***listas enlazadas*** son un conjunto de estructuras auto referenciadas llamadas ***nodos***. Podemos guardar y modificar datos en el momento de ejecución, sin ser necesario decir cuántos espacios tendrá nuestra lista.

Los ***nodos*** tienen al menos uno de los campos de la estructura como puntero de una estructura del mismo tipo. La forma de un nodo es la siguiente:

**struct** nodo

{

**char** \*nombre;

**struct** nodo \*siguiente;

};

Un ***nodo*** en c es una estructura que se crea con memoria dinámica, podemos encontrarnos ***typedef*** que nos permite crear sinónimos para nuestra estructura.

**typedef struct** nodo

{

**char** \*nombre;

**struct** nodo \*siguiente;

};

Los miembros de la estructura Libro son:

• content: la información contenida por el nodo.

void \*: permite guardar cualquier tipo de información.

• next: la dirección del siguiente nodo, o NULL si el siguiente nodo es el último.

***va\_list***

No es una funcion, es una estructura que será usada por el resto de funciones permitidas: va\_start(), va\_arg(), y va\_end().

Prototipo:

va\_list variable\_name;

***va\_start***

Esta funcion no devuelve nada, pero tiene dos parámetros. El primero va\_list ya declarada probablemente y el segundo el parámetro el menos conocido.

Va\_start inicializará el argument\_list usando las funciones va\_arg y va\_end.

**Prototipo:**

**void** va\_start (va\_list argument\_list, last\_known\_parameter);

***va\_arg***

No es una funcion, es una estructura que será usada por el resto de funciones permitidas: va\_start(), va\_arg(), y va\_end().

Prototipo:

va\_arg variable\_name;

***va\_end***

No es una funcion, es una estructura que será usada por el resto de funciones permitidas: va\_start(), va\_arg(), y va\_end().

Prototipo:

Va\_end variable\_name;