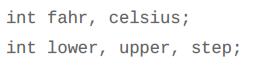
Capítulos: 1.2, 1.3, 1.4, 1.5, 3

### 1.2 Variables y expresiones aritméticas

En C se deben declarar todas las variables antes de su uso, generalmente al principio de la función y antes de cualquier proposición ejecutable. Una declaración notifica las propiedades de una variable; consta de un nombre de tipo y una lista de variables, como



### 1.3 La proposición for

La proposición **for** es un ciclo. Dentro de los paréntesis existen tres secciones, separadas por punto y coma.

La primera, la **inicialización** se ejecuta una vez, antes de entrar propiamente al ciclo.

La segunda sección es la **condición** o **prueba** que controla el ciclo. Esta condición se evalúa; si es verdadera, el cuerpo del ciclo se ejecuta.

La última es el incremento de **avance.**

El ciclo termina si la condición se hace falsa. Tal como con el while, el cuerpo del ciclo puede ser una proposición sencilla o un grupo de proposiciones encerradas entre llaves.

La selección entre while y for es arbitraria y se basa en aquello que parezca más claro. El for por el general es apropiado para ciclos en los que la inicialización y el incremento son proposiciones sencillas y lógicamente relacionadas, puesto que es más compacto que el while y mantiene reunidas en un lugar a las proposiciones que controlan el ciclo.

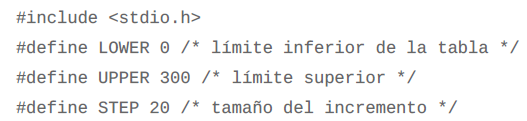
### 1.4 Constantes Simbólicas

Es una mala práctica poner “números mágicos”, ya que proporcionan muy poca información a quien tenga que leer el programa, y son difíciles de modificar de una forma sistemática.

Una manera de tratar a esos números mágicos es darles nombre significativos. Una línea **#define** define un nombre simbólico o constante simbólica como una cadena de caracteres especial.



A partir de esto, cualquier ocurrencia de nombre se sustituirá por el texto de reemplazo correspondiente.

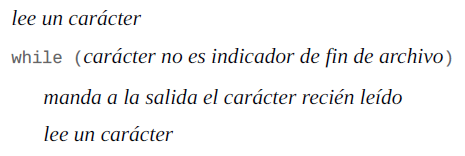


Los nombres de constantes simbólicas, por convención, se escriben en mayúsculas, de modo que se puedan distinguir fácilmente de nombres de variables.

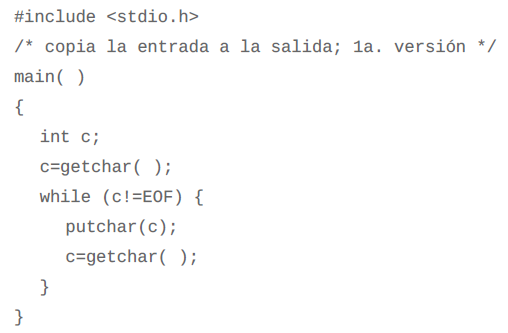
**Nótese que no hay punto y coma al final de una línea #define.**

### 1.5 Entrada y salida de caracteres

#### 1.5.1 Copia de archivos



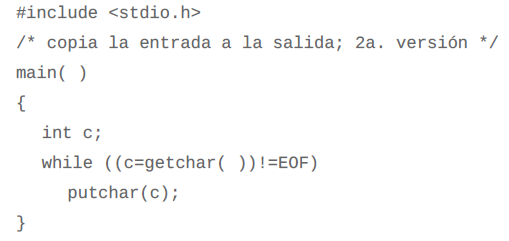
Al convertir esto en C se obtiene



Usamos **int** por una sutil pero importante razón. El problema es distinguir el fin de la entrada de datos válidos. La solución es que getchar devuelva un valor distintivo cuando no hay más a la entrada, un valor que no puede ser confundido con ningún otro carácter. Este valor se llama **EOF** (end of file).

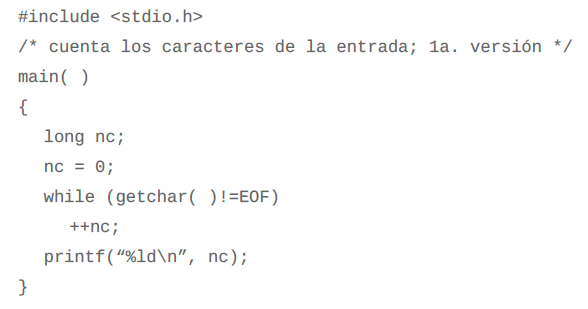
Se debe declarar c con un tipo que sea lo suficientemente grande para almacenar cualquier valor que le regrese getchar. No se puede utilizar char puesto que c debe suficiente para mantener a EOF además de cualquier otro carácter. Por lo tanto se emplea **int**.

EOF es un valor numérico definido en <stdio.h>, pero el valor numérico específico no importa mientras que no sea el mismo que ningún valor tipo char. Utilizando la constante simbólica, hemos asegurado que nada en el programa depende del valor numérico específico.

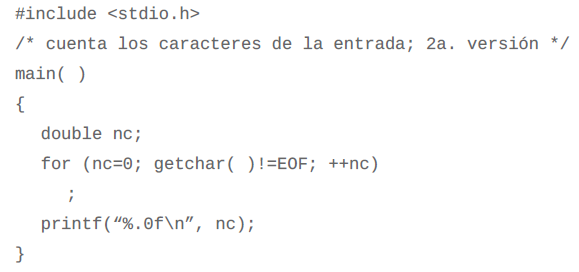


#### 1.5.2 Conteo de caracteres

El siguiente programa cuenta caracteres y es semejante al programa que copia



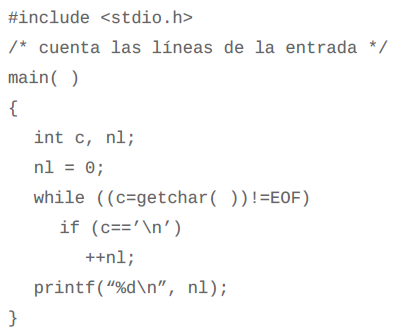
Otra forma de imprimir el ciclo



Uno de los aspectos agradables de while y for es que hacen la prueba al inicio del ciclo, antes de proceder con el cuerpo.

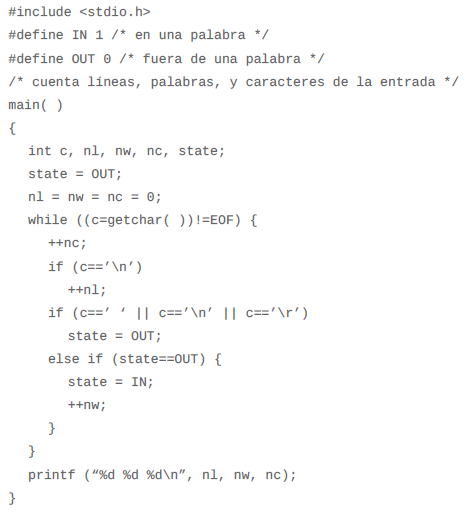
#### 1.5.3 Conteo de líneas

El siguiente programa cuenta líneas a la entrada



Un carácter escrito entre apóstrofos representa un valor entero igual al valor numérico del carácter en el conjunto de caracteres de la máquina. Por ejemplo, ‘A’, en el conjunto ASCII equivale a 65.

#### 1.5.4 Conteo de palabras



### 

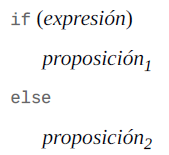
### 3.1 Proposiciones y Bloques

En C, el punto y coma es un terminador de proposición.

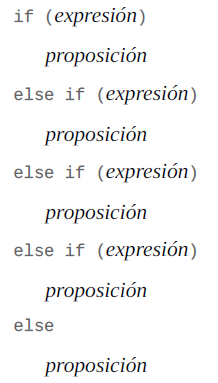
Las llaves { y } se emplean para agrupar declaraciones y proposiciones dentro de una proposición compuesta o bloque.

### 3.2 if-else

La proposición **if-else** se utiliza para expresar decisiones. Formalmente, la sintaxis es

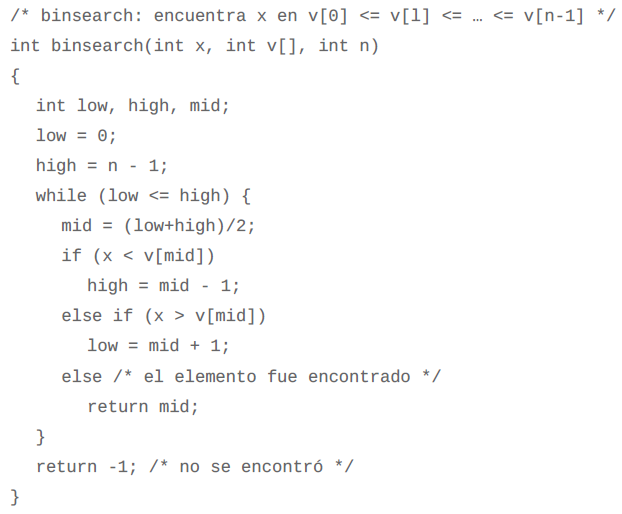


### 3.3 else-if



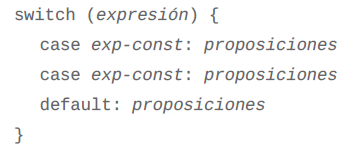
Esta secuencia de proposiciones if es la forma más general de escribir una decisión múltiple. Las expresiones se evalúan en orden; si cualquier expresión es verdadera, la proposición asociada con ella se ejecuta, y esto termina toda la cadena.

La parte del último else maneja el caso “ninguno de los anteriores” o caso por omisión cuando ninguna de las otras condiciones se satisface. En algunos casos no hay una acción explícita para la omisión; en ese caso, puede omitirse; o utilizarse para detección de errores al atrapar una condición “imposible”.



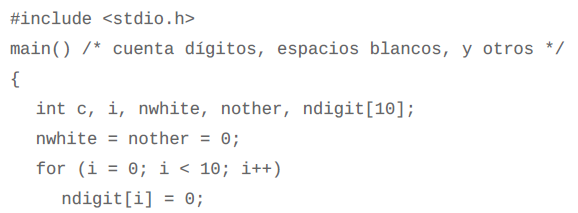
### 3.4 switch

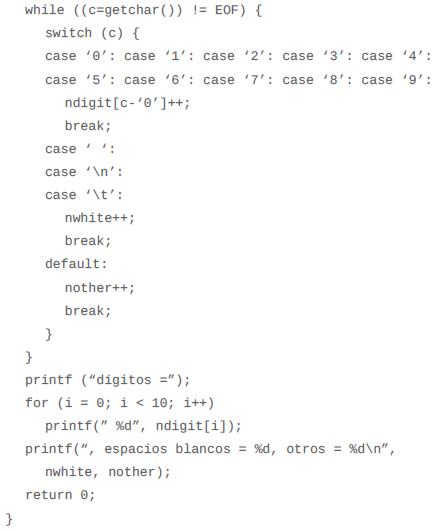
La proposición **switch** es una decisión múltiple que prueba si una expresión coincide con uno de un número de valores constantes enteros, y traslada el control adecuadamente.



Cada **case** se etiqueta con uno o más valores constantes enteros o expresiones constantes enteras. Si un case coincide con el valor de la expresión, la ejecución comienza allí. Todas las expresiones case deben ser diferentes.

El etiquetado como **default** se ejecuta si ninguno de los otros se satisface. El default es optativo.





La proposición **break** provoca una salida inmediata del switch. Puesto que los case sólo sirven como etiquetas, después de que se ejecute el código para uno, la ejecución pasa al siguiente, a menos que se toma una acción específica para terminar el switch.

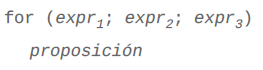
### 3.5 Ciclos -- while y for

Proposición **while**

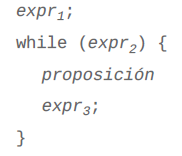


En la expresión se evalúa. Si es diferente de cero, se ejecuta la proposición y se reevalúa la expresión. Este ciclo continúa hasta que la expresión se hace cero.

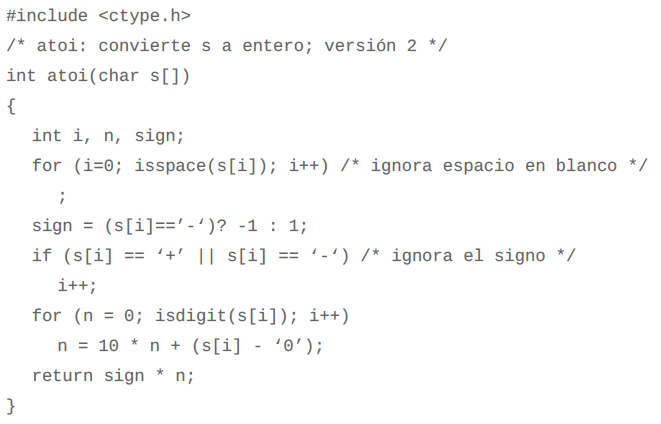
La proposición **for**

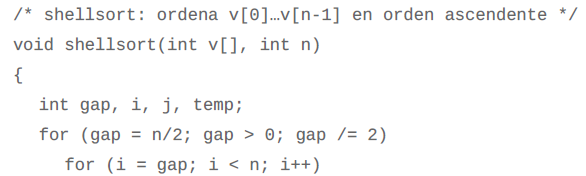
****

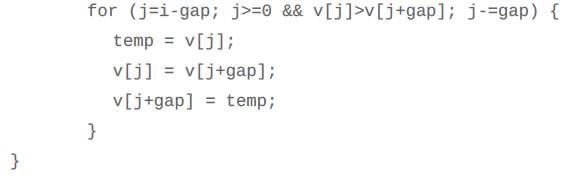
es equivalente a

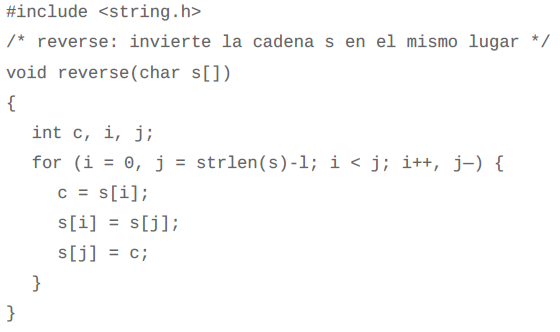


Función **atoi**, que convierte una cadena a su equivalente numérico.



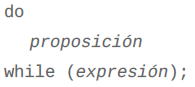




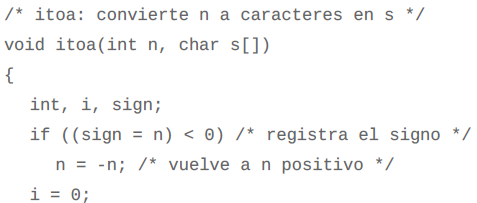


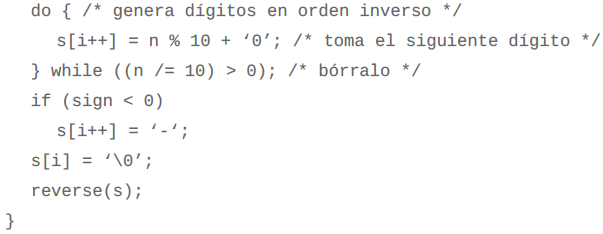
### 3.6 Ciclo -- do-while

El ciclo **do-while** prueba al final después de realizar cada paso a través del cuerpo del ciclo, el cual se ejecuta siempre por lo menos una vez.



La proposición se ejecuta y después se evalúa la expresión. Si es verdadera, la proposición se evalúa de nuevo, y así sucesivamente. Cuando la expresión se hace falsa, el ciclo termina.





### 3.7 break y continue

### 3.8 goto y etiquetas