Conceptos básicos de la informática

Computadora y cálculos. Estructura

La **informática** según la RAE es

> Conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información por medio de los ordenadores.

- (Informática = INFORmación + autoMÁTICA)

En literatura anglosajona

- Computer Science, ciencia de los computadores
- Computer Engineering
- IT Engineering

En literatura francófona

- Informatique

La **programación** según la RAE es

> Acción o efecto de programar

Programar según la RAE es:

1. Formar programas, previa declaración de lo que se piensa hacer y anuncio de las partes de que se ha de componer un acto o espectáculo o una serie de ellos.
2. Idear y ordenar las acciones necesarias para realizar un proyecto.
3. **Preparar ciertas máquinas o aparatos para que empiecen a funcionar en el momento y en la forma deseados.**
4. **Elaborar programas para su empleo en computadoras.**

Máquinas programables

Una **máquina** es un cierto dispositivo físico capaz de realizar un cierto trabajo u operación.

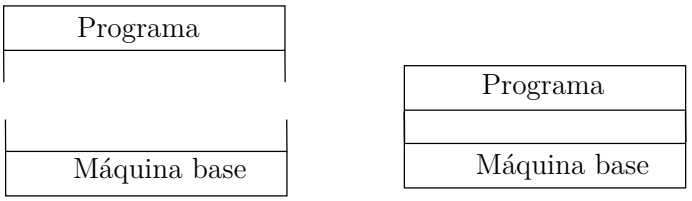
Este concepto puede extenderse a cuando consideramos máquinas que no existen físicamente, pero en las que puede describirse y concebirse su comportamiento, se denominan **máquinas virtuales**.

Atendiendo a como funcionan se pueden clasificar las máquinas en:

- Máquinas no automáticas o de control manual. Es preciso un agente externo u operador que desencadena las operaciones.

- Máquinas automáticas. Actúan por si solas, sin necesidad de operador, pero pueden reaccionar ante estímulos externos.

Existen otras máquinas automáticas denominadas **máquinas programables**, cuyo comportamiento fijo se completa con una parte, *programa*, que permite modificar el comportamiento de la máquina base. Tal como es muestra en la siguiente figura:



Basta con cambiar el programa para tener una nueva máquina con un comportamiento diferente.

Veamos un ejemplo sacado del mundo de los instrumentos musicales:

- Máquina no automática (requiere intervención manual constante): El órgano tradicional (como el órgano de tubos) es un buen ejemplo. Para que funcione, necesita que una persona accione los teclados y pedales constantemente, además de controlar el flujo de aire. Necesitando dos usuarios: el calcante tenía que bombear aire de manera constante y precisa para que el organista pudiera tocar. Con la llegada de motores eléctricos, los órganos dejaron de necesitar una segunda persona para accionar los fuelles, ya que el aire comenzó a ser proporcionado automáticamente por estos sistemas.
- Máquina automática (funciona sin intervención humana directa durante su operación, pero no es programable): La caja de música es un gran ejemplo. Una vez que se da cuerda, la máquina reproduce la música de manera automática, siguiendo un patrón fijo que no se puede modificar, que esta definido en el momento que se construye la caja de música.
- Máquina automática programable (puede programarse para realizar diferentes acciones sin intervención humana durante su operación): El organillo o pianola es un buen ejemplo de máquina automática programable. Estos instrumentos funcionan automáticamente siguiendo una secuencia predefinida, pero lo interesante es que se les puede cambiar la "programación" (rollos de papel perforado en la pianola o cilindros en el organillo) para reproducir diferentes melodías.

La diferencia clave entre cada tipo de máquina en cuanto a intervención humana y capacidad de programación. Por ejemplo el organillo tradicional (o órgano de manivela) es programable pero no es automático necesita al organillero que gire la manivela para que funcione. La pianola sí es completamente automática en cuanto a la ejecución de la música. Una vez que se coloca el rollo de papel perforado y se acciona, la pianola reproduce la música sin necesidad de intervención humana constante.

Por lo tanto, aunque ambos pueden considerarse máquinas programables, el organillo requiere intervención física del operador, mientras que la pianola puede funcionar sin que una persona esté continuamente involucrada durante la ejecución de la música.

Concepto de cómputo

La definición de diccionario de cómputo nos dice que es: *Cálculo para averiguar el resultado, el valor de algo.*

Un cómputo por tanto implica el *tratamiento de información*, numérica de forma tradicional, pero se puede extrapolar a cualquier otro tipo de información.

Cualquier computo se puede describir de diferentes maneras. Por ejemplo el computo para **Calcular la nota final** de un alumno que ha realizado tres exámenes a lo largo del curso, pero sabiendo que si en uno de los exámenes tiene un cero entonces su nota es cero:

- Una opción es utilizar una fórmula

$Nota \approx \sqrt[3]{nota_1 * nota_2 * nota_3}$

- Pero también se puede describir como un proceso:

```
1. Nota = 0;
2. Si nota1 no es cero Nota = Nota + nota1
   - Sino  Nota = 0 e ir a paso 5
3. Si nota2 no es cero Nota = Nota + nota2
   - Sino  Nota = 0 e ir a paso 5
4. Si nota3 no es cero Nota = Nota + nota3
   - Sino  Nota = 0 e ir a paso 5
5. Nota = Nota / 3
```

- O incluso una expresión lógica. Si el resultado de esta expresión es cierto entonces la nota es 0.

$Nota = \neg(((nota_1 + nota_2 + nota_3) = (nota_1 + nota_2) \vee ((nota_1 + nota_2 + nota_3) = (nota_1 + nota_3)) \vee ((nota_1 + nota_2 + nota_3) = (nota_2 + nota_3)))$

Concepto de computador

Un **computador** es una máquina programable para el tratamiento de información, es decir realización de cómputos. Posee unos elementos **fijos** o máquina de base que llamamos **hardware** y otros modificables que son los programas o **software**.

Los computadores actuales son máquinas con programa almacenado de forma que la modificación de los programas no implica la modificación de los elementos físicos y por tanto necesitamos sistemas de almacenamiento y de comunicación del ordenador con el entorno (recordemos los discos de una pianola).

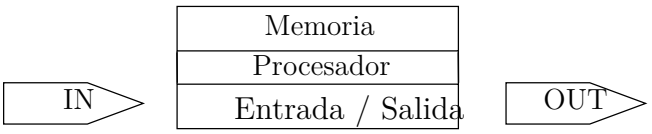
Así, una versión más descriptiva del concepto de computadora donde se indica el proceso seguido para las tareas de cómputo sería:

Un **computador o computadora** es una máquina formada por elementos de tipo **electrónico**, capaces de aceptar unos datos de **entrada**, realizar con ellos gran variedad de tareas (**operaciones**) y proporcionar la información resultante a través de un medio de **salida**, bajo el control de un **programa**, previamente **almacenado** en el propio computador. Esto puede incluir tanto dispositivos físicos como servidores, laptops, tablets, entre otros, así como también máquinas virtuales desplegadas en la nube.

Las **máquinas virtuales** en la nube son entornos computacionales virtuales que se ejecutan en infraestructuras de computación remota y están disponibles a través de Internet. Estas máquinas virtuales proporcionan capacidades informáticas similares a las de una computadora física, pero con la flexibilidad y escalabilidad adicionales ofrecidas por la computación en la nube.

Un **programa** será por lo tanto la descripción de un cómputo en un lenguaje de programación, que como se ha visto en el ejemplo de la nota media puede tener muchas formas o lenguajes diferentes, aunque nosotros nos centraremos en la definición de programas como un conjunto de **instrucciones**, lo que llamamos **programación estructurada**, sobre el ejemplo del cómputo de la nota media es una notacion en forma de proceso.

Gráficamente esta última definición de computadoras se recoge en la siguiente imagen



La memoria almacena datos y programas. Los dispositivos de Entrada/Salida permiten intercambiar información con el exterior. El procesador es el elemento de control que realiza las operaciones elementales del tratamiento de la información interna y de las operaciones de entrada/salida de la información al exterior (de acuerdo con los programas almacenados en la memoria). Es usual llamar **periféricos** a los dispositivos electrónicos que conforman las unidades de entrada y de salida.

Estructura de una computadora

Cada año, probablemente esperamos pagar al menos un poco más por la mayoría de los productos y servicios, sube la luz, el agua, las matrículas. Lo contrario ha ocurrido en los campos de la informática y las comunicaciones, especialmente en lo que se refiere a la electrónica que soporta estas tecnologías. A lo largo de los años los costes del hardware han disminuido rápidamente. Durante décadas, cada par de años, la potencia de procesamiento de los ordenadores aproximadamente es el doble. Esta notable tendencia suele denominarse **Ley de Moore**, llamada así por Gordon Moore, cofundador de Intel 1960.

En el campo de las comunicaciones se ha producido un crecimiento similar. Los costes se han desplomado a medida que la enorme demanda de ancho de banda de las comunicaciones (es decir, la capacidad de transmisión de información) ha traído una intensa competencia.

Los componentes electrónicos de una computadora son cada día más complejos, y muchos de ellos incluso se duplican. Existen numerosas combinaciones de dispositivos/componentes físicos que pueden ensamblarse para crear computadoras. Lo que es más, actualmente aparece el concepto de **computadores virtuales** o **máquinas virtuales**, que no necesitan *hardware* como tal. Se definen o alquilan componentes/unidades lógicas sobre servidores remotos para crear computadoras personalizadas virtuales. Es decir, la máquina base se define de forma virtual, definiendo las unidades lógicas que se despliegan sobre soportes físicos de distinta naturaleza.

Independientemente de las diferencias físicas, los ordenadores combinan distintos tipos de **unidades lógicas** o secciones: (Unidades de entrada, de salida, de memoria, de control, aritmético lógica y de memoria secundaria o almacenamiento)

Unidad de entrada

Esta unidad obtiene la información (datos y programas informáticos) de los dispositivos de entrada y la pone a disposición de las demás unidades para su procesamiento. Los computadores reciben la mayoría de las entradas de los usuarios a través de teclados, pantallas táctiles, ratones y touchpads.

Otras formas de entrada son:

- la recepción de comandos de voz,
- escanear imágenes y códigos de barras,
- leer datos de dispositivos de almacenamiento secundarios (como unidades de discos duros, Blu-ray Disc™ y memorias USB, también llamados "lápices de memoria"),
- recibir vídeo de una cámara web,
- recibir información de Internet (por ejemplo, cuando transmite vídeos de de YouTube® o descarga libros electrónicos de Amazon),
- recibir datos de posición de un dispositivo GPS,
- recibir información de movimiento y orientación de un acelerómetro (un dispositivo que responde a la aceleración hacia arriba/abajo, izquierda/derecha y adelante/atrás).
- recibir la voz de asistentes inteligentes como Apple Siri, Amazon Alexa y Google Home.

Unidad de salida

Esta unidad "envía" la información que el computador ha procesado y la coloca en varios dispositivos de salida para que esté disponible fuera del computador. La mayor parte de la información que sale de los ordenadores hoy en día:

- se muestra en pantallas,
- se imprime en papel ("ser verde" desaconseja esto),
- se reproduce en audio o vídeo en teléfonos inteligentes, tabletas, ordenadores y pantallas gigantes en estadios deportivos,
- se transmite por Internet o
- para controlar otros dispositivos, como coches que se conducen solos (y vehículos autónomos en general), robots y electrodomésticos "inteligentes".

También es habitual que la información se transmita a dispositivos de almacenamiento secundario, como las unidades de estado sólido (SSD), los discos duros, memorias USB y unidades de DVD. Son unidades donde se hace **persistente** la información, es decir, no se pierde aunque se apague la máquina.

Otras formas de salida son la vibración de teléfonos inteligentes, mandos de juegos y dispositivos de realidad virtual.

Unidad de memoria

Esta unidad es un "almacén" de acceso rápido y capacidad relativamente baja que mantiene la información introducida a través de la unidad de entrada, haciéndola disponible cuando sea necesario. La unidad de memoria también guarda la información procesada hasta que puede ser colocada en dispositivos de salida.

La información de la unidad de memoria es volátil. Se pierde cuando se apaga el ordenador. La unidad de memoria suele denominarse memoria principal, memoria primaria o RAM (Random Access Memory).

Se divide en posiciones (**palabras de memoria**) de un determinado tamaño. Se accede a cada posición a través de un número (**dirección de memoria**).

Las memorias principales de los ordenadores de sobremesa y portátiles contienen hasta 128 GB de RAM, aunque lo más habitual es entre 8 y 16 GB. 16 GB es lo más habitual. GB son las siglas de gigabytes; un gigabyte equivale aproximadamente a mil millones de bytes.

Un byte son ocho bits. Un bit (abreviatura de "dígito binario") es un 0 o un 1. EstoEstos conceptos se verán más adelante. Son las unidades en las que se mide la capacidad de almacenamiento o velocidad de trasmisión de información, son por tanto unidades para medir la cantidad de información.

Parámetros significativos de la memoria:

- Tamaño de la memoria expresado en Megabytes (1 GigaByte = 1024 MegaBytes; 1 MegaByte = 1024 Kilobytes; 1 Kilobyte = 1024 Bytes; 1 Byte = 8 bits; bit = dígito binario: 0 ó 1).
- Tiempo de acceso expresado en nanosegundos (1 ns = 10-9 segundos).
- Ancho de banda expresado en Megabytes transferidos a ó desde la memoria por segundo.

Unidad Central aritmético lógica (ALU)

Esta unidad "fabrica", es decir, realiza cálculos (por ejemplo, suma, resta, multiplicación y división) y toma decisiones (por ejemplo, comparar dos elementos de la unidad de memoria para determinar si son iguales). En los sistemas actuales, la ALU forma parte de la siguiente unidad lógica, la CPU.

Unidad central de proceso

Esta unidad es la "administrativa" coordina y supervisa el funcionamiento de las demás unidades, incluyendo habitualmente el procesador y la unidad aritmético lógica (ALU). La CPU indica:

- a la unidad de entrada, cuándo leer información en la unidad de memoria,
- a la ALU cuándo utilizar la información de la unidad de memoria en los cálculos, y
- a la unidad de salida cuándo enviar información desde la unidad de memoria a dispositivos de salida,
- cuando y como se accede a la memoria.

La mayoría de los ordenadores actuales tienen procesadores multinúcleo que implementan de forma económica múltiples procesadores en un único chip de circuito integrado. Estos procesadores pueden realizar muchas operaciones simultáneamente. Un procesador de doble núcleo tiene dos CPU, un procesador de cuatro núcleos tiene cuatro y un procesador octa-core tiene ocho. Intel tiene algunos procesadores con hasta 72 núcleos.

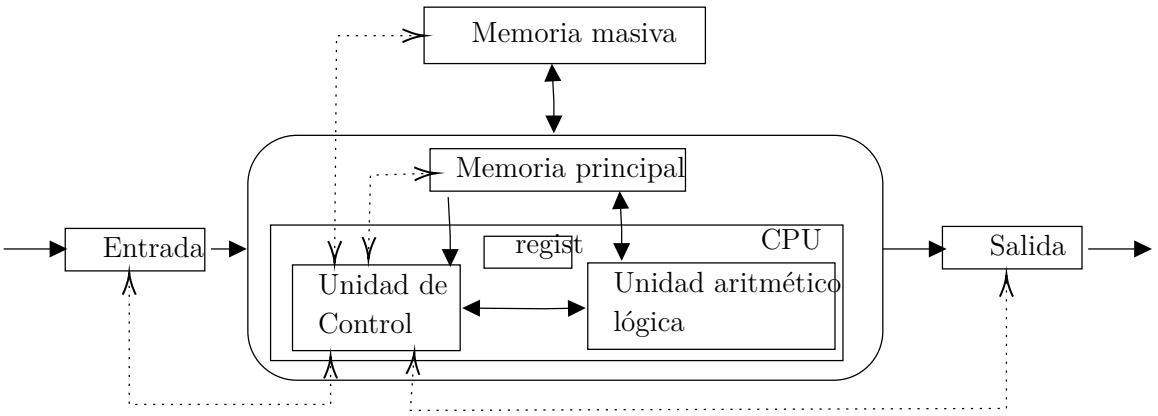
Unidad de almacenamiento secundario

Es la unidad de "almacenamiento" a largo plazo y de gran capacidad, siendo tanto de entrada como de salida. Los programas y datos utilizados activamente por las otras unidades se colocan en dispositivos de almacenamiento secundario (también llamada memoria secundaria) hasta que vuelvan a necesitarse, posiblemente horas, días, meses o incluso años después. También se llama memoria masiva.

La información en el almacenamiento secundario es persistente: se conserva incluso cuando el computador se apaga. Se tarda mucho más en acceder a la información del almacenamiento secundario que información de la memoria primaria, pero su coste por byte es mucho menor.

Ejemplos de dispositivos de almacenamiento secundarios son las unidades de estado sólido (SSD), las memorias flash USB, los discos duros y las unidades Blu-ray de lectura/escritura.

Muchas unidades actuales almacenan terabytes (TB) de datos. Un terabyte es aproximadamente un billón de bytes. Los discos duros de ordenadores de sobremesa y portátiles que tienen una capacidad de hasta 4 TB, y algunos discos duros recientes para ordenadores de sobremesa tienen una capacidad de hasta 20 TB.



Otros elementos que completan la arquitectura clásica de un ordenador son:

Buses: Conjuntos o grupos de hilos que interconectan las unidades de procesamiento y de control y la memoria, así como estas con los periféricos (unidades de entrada y salida), proporcionando un camino de comunicación para el flujo de datos entre las distintas unidades. La información se transmite en paralelo (en un instante de tiempo dado se encuentran en el bus todos los bits de un dato).

Registros: Dispositivos físicos que tienen una labor específica o que permiten la realización de operaciones por parte de la CPU. Los más importantes son:

- registro de instrucción (IR) almacena la instrucción que se está ejecutando

- contador de programa (PC) contiene la dirección de memoria de la siguiente instrucción a ejecutar
- puntero de pila (SP)
- r0-rD: registros de uso general (RF: banco de registros)

Representación de la información

De las definiciones de **dato** según la RAE, son del contexto de la computación la 1 y la 3. En concreto la tercera es sólo relativa a la informática.

1. m. Información sobre algo concreto que permite su conocimiento exacto o sirve para deducir las consecuencias derivadas de un hecho. A este problema le faltan datos numéricos.
2. m. Documento, testimonio, fundamento.
3. m. *Inform.* Información dispuesta de manera adecuada para su tratamiento por una computadora.

Siendo más específicos un **Dato** en informática es: una representación formalizada de hechos o conceptos susceptible de ser comunicada y/o procesada.

Existen distintos **tipos de datos** que por lo tanto tienen que ser representados de forma diferente:

- Numéricos (12, 28.5): reales o enteros
- Alfabéticos (Ana)
- Alfanuméricos: 23456X, M-6995
- Imágenes, sonido, video,...

Llamamos *representación de la información* a la forma en la que se almacenan y recogen los datos para poder ser procesados en la computadora. Se debe tener en cuenta que en los medios electrónicos de almacenamiento / procesamiento solo disponen de **dos estados**: (con permiso de las tecnologías cuánticas)

- interruptor (relé) abierto-cerrado
- paso o no paso de corriente
- magnetización en un sentido u otro

Esto se traslada a que la información procesada por un computador son siempre 1s y 0s. Lo que lleva a la necesidad de **traducir** cualquier dato a una combinación de esos dos símbolos es decir **notación binaria** o sistema de numeración binario.

Sistemas de numeración en informática

Los sistemas de numeración usuales en informática no son decimales, no utilizan la base 10 como los humanos.

Tipo	Base	Uso	Alfabeto
decimal	10	humanos	0 a 9
binario	2	computadoras	0, 1
octal	8	computadoras (códigos intermedios)	0 a 7
hexadecimal	16	computadoras (códigos intermedios)	0 a 9, A, B, C, D, E, F

De forma general un sistema de numeración base b: (Sistema de numeración posicional)

Alfabeto tiene **b** símbolos o cifras diferentes, con lo que un Número = {cifras}

Su valor depende de:

- Nº de cifras
- Valor de cada cifra
- Posición cifra dentro del número

Para el sistema de numeración decimal

$\text{Alfabeto}_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Ejemplo de número decimal:

$4325,12 = 4000 + 300 + 20 + 5 + 0,1 + 0,02 = 4 * 10^3 + 3 * 10^2 + 2 * 10^1 + 5 * 10^0 + 1 * 10^{(-1)} + 2 * 10^{(-2)}$

Generalización para cualquier base b:

$\text{Alfabeto}_b = \{0, 1, 2, ..., b-1\}$

$N = \ldots\ n_4 n_3 n_2 n_1 n_0 n_{-1} n_{-2} n_{-3}\ \ldots = \ldots + n_4 * b^4 + n_3 * b^3 + n_2 * b^2 + n_1 * b^1 + n_0 * b^0 + n_{-1} * b^{-1} + n_{-2} * b^{-2} + \ldots$

Sistema de numeración base 2: .

Solo hay

1 0

tipos de personas:

Las que entienden binario
y las que no.

.

.

Nº decimal	Nº binario
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

Observando la tabla se comprueba que por ejemplo para representar el "6" en decimal, necesitamos 3 posiciones en binario. Cada una de estas posiciones es un **bit**.

Por tanto si se quieren representar en binario las posibles estados emocionales de una persona:

Estados= { Tranquilo, Feliz, Emocionado, Preocupado, Triste, Enojado, Sorprendido, Nervioso, Cansado, Motivado }

Son necesarios cuatro cifras binarias. 2^4 es el máximo número de elementos a representar, en este caso tenemos 10 estados.

BIT

- Los dígitos en el sistema de numeración binario se llaman bits (BIT= Binary digit).
- Es la unidad más pequeña de información, un uno o un cero.

Byte

- Es un conjunto de 8 bits.
- También se llama Octeto o Carácter (porque con un byte se suele codificar un carácter).

Con 1 bit se pueden representar hasta 2^1 elementos {0,1}

Con 2 bits se pueden representar hasta 2^2 elementos {00, 01, 10, 11}

Con 8 bits podemos representar hasta 256 elementos 2^8

```
0000 0000
```

hasta

```
1111 1111
```

Pregunta: En un lago hay una superficie cubierta de nenúfares y cada día esa extensión dobla su tamaño. Si tarda 48 días en cubrir el lago, ¿cuánto tarda en cubrir la mitad del lago?

Esta es la razón por la que la unidad de almacenamiento de información en informática (tamaño memoria, tamaño almacenamiento secundario, registros y buses) se mide en potencias de 2.

Múltiplos del byte

acrónimo	nombre	equivalencia
KB	Kilobyte	2^{10} bytes = 1024 bytes $\approx 10^3$ bytes
MB	Megabyte	2^{20} bytes = 1048576 bytes $\approx 10^6$ bytes
GB	Gigabyte	2^{30} bytes = 1073741824 bytes $\approx 10^9$ bytes
TB	Terabyte	2^{40} bytes $\approx 10^{12}$ bytes
PB	Petabyte	2^{50} bytes $\approx 10^{15}$ bytes
EB	Exabyte	2^{60} bytes $\approx 10^{18}$ bytes

qbits (cúbit)

Unidad relacionada con la computación cuántica. Los **qubits** pueden estar en una superposición de estados *{0, 1}*, lo que significa que pueden representar simultáneamente 0 y 1. Además, el estado de un qubit depende del estado de otro, lo que proporciona una potencialmente mayor capacidad de procesamiento y almacenamiento de información en comparación con los bits clásicos.

Aritmética del binario Lo primero es saber como trasformar número entre bases

De binario a decimal

$(110100)_2 = (1 * 2^5) + (1 * 2^4) + (1 * 2^2) = 32 + 16 + 4 = 52$

De decimal a binario

Parte entera: dividir sucesivamente por 2 y utilizar el último cociente y los restos de las divisiones para las cifras binarias (el último cociente es el bit más significativo y el primer resto es el bit menos significativo):

$$
\begin{array}{r|l}
26 & 2 \\
\hline
06 & 13 \;\; 2 \\
\hline
0 & 1 \;\; 6 \;\; 2 \\
\hline
 & 0 \;\; 3 \;\; 2 \\
\hline
 & 1 \;\; 1
\end{array}
$$

$(26)_{10} = (11010)_2$

Parte fraccionaria: multiplicar sucesivamente por 2 las partes fraccionarias resultantes y utilizar las partes enteras en orden para las cifras binarias:

$$\begin{array}{r} 0.1\,8\,7\,5 \\ \times \quad 2 \\ \hline 0.3\,7\,5\,0 \end{array} \quad \begin{array}{r} 0.3\,7\,5 \\ \times \quad 2 \\ \hline 0.7\,5\,0 \end{array} \quad \begin{array}{r} 0.7\,5 \\ \times \quad 2 \\ \hline 1.5\,0 \end{array} \quad \begin{array}{r} 0.5 \\ \times \quad 2 \\ \hline 1.0 \end{array}$$

$(0,1875)_{10} = (0,0011)_2$

Operaciones en binario

a	b	a + b	a – b	a * b	a / b
0	0	0	0	0	indeterminado
0	1	1	1 y me adeudo 1	0	0
1	0	1	1	0	
1	1	0 y me llevo 1	0	1	1

Representación en complementos

Útiles para representar números negativos (se reducen las restas a sumas y se simplifica el hardware)

- Complemento **a la base-1** de un número N es el número que resulta de restar cada una de las cifras de N a la base menos 1 del sistema de numeración que se esté utilizando. Se obtiene cambiando 0 por 1 y 1 por 0.
- Complemento **a la base** de un número N es el número que resulta de restar cada una de las cifras de N a la base menos uno del sistema de numeración que se esté utilizando, y posteriormente sumar 1 a la diferencia obtenida.

Ejemplos:

Numero	Base	Complemento a base-1	Complemento a base
63	10	36	37
10010	2	01101	01110

Como calcular complementos en binario

- Complemento a 1: Cambiar los ceros por unos y los unos por ceros
- Complemento a 2: de derecha a izquierda se deja igual hasta el primer 1 y se cambian a partir de este.

Codificación de datos numéricos enteros

Las principales formas de representar los enteros son en **binario**, con o sin signo y utilizando el decimal codificado en binario (**BCD**)

BCD (decimal codificada en binario)

4 bits para representar las cifras: 0 1 2 3 4 5 6 7 8 9

Dado el número:98325

```
 9       8     3       2     5
1001   1000   0011   0010   0101
```

BINARIA

- Un numero entero puede ser representado en base dos (sistema de numeración binario)

  $25 = 16 + 8 + 1 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 11001$

- Módulo y signo
  - Se usa un bit adicional para representar el signo.
- Representación en complemento
  - Es más apropiada para el procesamiento de la información en un ordenador.

De esta forma tenemos:

Enteros sin signo

Con n bits tenemos 2^n números

Rango: $[0, 2^n - 1]$

Enteros con signo

Cuando se utiliza el bit más significativo para el signo se puede optar por tres alternativas, mantener la magnitud en binario o utilizar complemento a 1 o complemento a 2

1. *Signo y magnitud*

- Primer bit: Recoge el signo (0 = positivo, 1 = negativo)
- Resto de bits: valor absoluto número

  Rango: $[-(2^{n-1} - 1), 2^{n-1} - 1]$

2. *Complemento a 1*

- Primer bit: signo (0=positivo, 1=negativo)
- Resto de bits:
  - Si el número es positivo: valor absoluto del número
  - Si el número es negativo: complemento a 1 de valor absoluto

  Rango: $[-(2^{n-1} - 1), 2^{n-1} - 1]$

3. *Complemento a 2*

- Primer bit: signo (0=positivo, 1=negativo)
- Resto de bits:
  - número positivo: valor absoluto del número

- número negativo: complemento a 2 de valor absoluto

Rango: $[-2^{n-1}, 2^{n-1} - 1]$

EJEMPLO:

Supongamos que tenemos 4 bits para representar los números, podremos representar 16 elementos 2^4. Si son enteros positivos podremos recoger del 0 al 15 simplemente pasándolos a binario. Si utilizamos el bit más significativo para el signo podremos representar $[-7, 7]$. Si se utiliza el complemento a 1 tendremos dos representaciones para el 0 cosa que no pasa con el complemento a dos.

nº Decimal	Signo y magnitud	Complemento a 1	Complemento a 2
+7	0111	0111	0111
+6	0110	...	...
+5	0101	...	
+4	0100		
+3	0011		
+2	0010		
+1	0001		
+0	0000		
-0	1000	1111	-
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8			1000

Codificación de datos numéricos reales

- Se representan por separado los tres elementos del número en notación científica: base, mantisa y exponente
  - Ejemplo: 0,123 x 10^{-4}
- Se representan por separado base (10), mantisa (0.123) y exponente (-4)
  - Permite representar rangos grandes de números usando pocos bits
- Actualmente se usa un sistema normalizado: norma IEEE 754
  - 32 bits
  - Base 2, está predeterminada, por lo que no es necesario codificarla
  - Signo: un bit, 0 = +, 1= -
  - Exponente: 8 bits (que utiliza una representación sesgada)

    - Mantisa: 23 bits

- Al representar de esta forma se pierde precisión

    - !Cuidado al comparar números! El ordenador sólo los considera iguales si todos los bits son iguales.
    - La norma IEEE 754 también contempla usar 64 bits (doble precisión)

- Enlace para ver la notación en bits de un número real
- Enlace a Conversor en línea
- Enlace para el estudio de las formas de representación de información numérica

Codificación de caracteres alfanuméricos

- Códigos de caracteres

    - **código** = correspondencia biunívoca entre un conjunto de caracteres y un conjunto de combinaciones de bits (tabla).
    - **caracteres**: numéricos, alfabéticos, alfanuméricos, especiales y de control.

- Código ASCII extendido

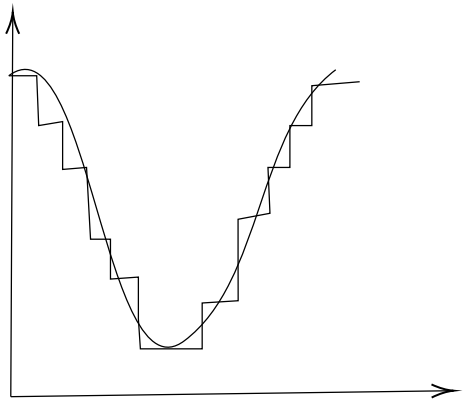
    - American Standard Code for Information Interchange
    - ASCII : solo 7 bits de los 8 que forman el byte
        - el bit más significativo (izquierda) no se utilizaba
        - Se pueden representar 2^7= 128 caracteres distintos
    - ASCII extendido: 8 bits
        - Con 8 bits (1 byte) se pueden representar $2^8 = 256$ caracteres diferentes, a continuación se muestran hasta 127

Decimal	8 bits	ASCII	Decimal	8 bits	ASCII	Decimal	8 bits	ASCII
0	00000000	[NUL]	1	00000001	[SOH]	2	00000010	[STX]
3	00000011	[ETX]	4	00000100	[EOT]	5	00000101	[ENQ]
6	00000110	[ACK]	7	00000111	[BEL]	8	00001000	[BS]
9	00001001	[TAB]	10	00001010	[LF]	11	00001011	[VT]
12	00001100	[FF]	13	00001100	[CR]	14	00001110	[SO]
15	00001111	[SI]	16	00010000	[DLE]	17	00010001	[DC1]
18	00010010	[DC2]	19	00010011	[DC3]	20	00010100	[DC4]
21	00010101	[NAK]	22	00010110	[SYN]	23	00010111	[ETB]
24	00011000	[CAN]	25	00011001	[EM]	26	00011010	[SUB]
27	00011011	[ESC]	28	00011100	[FS]	29	00011101	[GS]
30	00011110	[RS]	31	00011111	[US]	32	00100000	espacio
33	00100001	!	34	00100010	”	35	00100011	#
36	00100100	$	37	00100101	%	38	00100110	&
39	00100111	’	40	00101000	(	41	00101001	)
42	00101010	*	43	00101011	+	44	00101100	,

Decimal	8 bits	ASCII	Decimal	8 bits	ASCII	Decimal	8 bits	ASCII
45	00101101	-	46	00101110	.	47	00101111	/
48	00110000	0	49	00110001	1	50	00110010	2
51	00110011	3	52	00110100	4	53	00110101	5
54	00110110	6	55	00110111	7	56	00111000	8
57	00111001	9	58	00111010	:	59	00111011	;
60	00111100	<	61	00111101	=	62	00111110	>
63	00111111	?	64	01000000	@	65	01000001	A
66	01000010	B	67	01000011	C	68	01000100	D
69	01000101	E	70	01000110	F	71	01000111	G
72	01001000	H	73	01001001	I	74	01001010	J
75	01001011	K	76	01001100	L	77	01001101	M
78	01001110	N	79	01001111	O	80	01010000	P
81	01010001	Q	82	01010010	R	83	01010011	S
84	01010100	T	85	01010101	U	86	01010110	V
87	01010111	W	88	01011000	X	89	01011001	Y
90	01011010	Z	91	01011011	[	92	01011100	\
93	01011101	]	94	01011110	^	95	01011111	_
96	01100000	‘	97	01100001	a	98	01100010	b
99	01100011	c	100	01100100	d	101	01100101	e
102	01100110	f	103	01100111	g	104	01101000	h
105	01101001	i	106	01101010	j	107	01101011	k
108	01101100	l	109	01101101	m	110	01101110	n
111	01101111	o	112	01110000	p	113	01110001	q
114	01110010	r	115	01110011	s	116	01110100	t
117	01110101	u	118	01110110	v	119	01110111	w
120	01111000	x	121	01111001	y	122	01111010	z
123	01111011	{	124	01111100	|	125	01111101	}
126	01111110	~	127	01111111	[DEL]			

Codificación de sonido

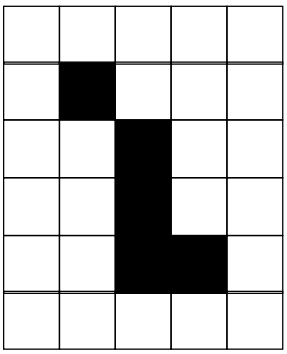
- El sonido es analógico
  - Hay que empezar por discretizarlo: muestreo (sampling)
  - Cada muestra se codifica (por ejemplo como un entero con un byte)
- Se pierde información
  - La calidad del sonido reconstruido dependerá de:
    - Número de muestras por segundo
    - Número de bits usados para codificar cada muestra



Codificación básica de Imágenes

-También es necesario muestrearla.

- División de la imagen en una matriz de píxeles
- A cada píxel se le asocia un valor (1 o 0, para blanco y negro, varios bits por píxel para imagen en color)

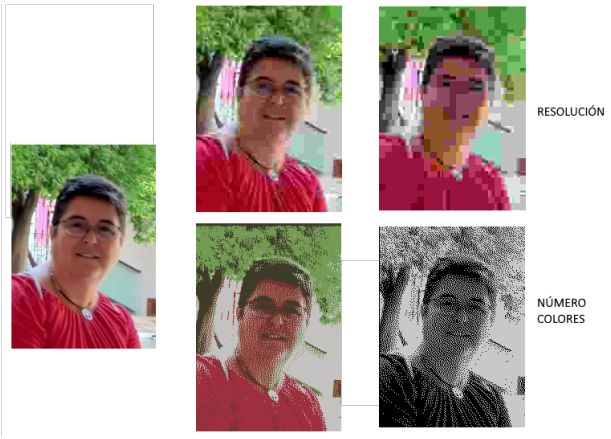


$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{matrix}$$

- Hay otras formas mejores (con compresión)

-La calidad de una imagen dependerá de:

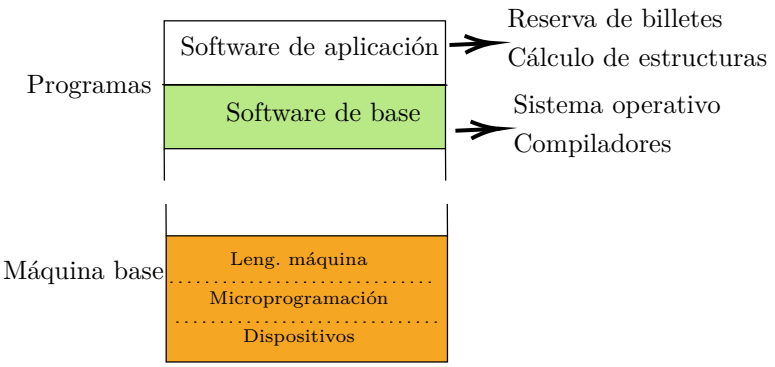
- Resolución: número de píxeles por unidad de superficie
- Número de bits por píxel (8,16,24,32)
    - 8 bits por píxel permite representar 256 colores diferentes

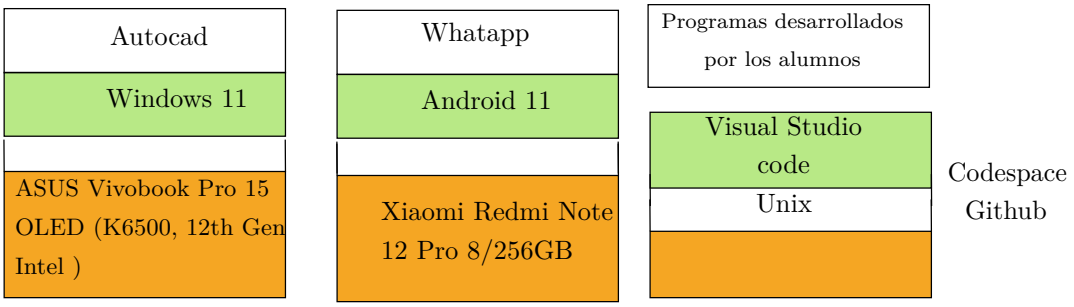


Algoritmos y lenguajes de programación

Veamos una versión extendida de la arquitectura de una computadora pero desde el punto de vista del software.

El Software es conjunto de programas asociados a una computadora que definen su comportamiento.





En esta imagen se destacan no los dispositivos físicos, puesto que los programas se almacenan en en la memoria (primaria y/o secundaria), sino que destacamos los distintos tipos de software que se pueden encontrar, y que se organizan por niveles.

- Software de base o programas del sistema: programas que gestionan el funcionamiento de la computadora:
    - *Sistema Operativo* - Pertenece a los componentes fijos de un máquina dada. Porque si bien es un elemento lógico, es necesario para que la máquina funcione. Un móvil o un portátil sin sistema operativo no es más que un bonito pisapapeles.
    - *Utilidades para construcción/ejecución de aplicaciones*. Destacamos aquí el concepto de **lenguaje máquina**; puesto que cada modelo de computadora física tiene una forma de entender los programas que depende de cuál sea su arquitectura y sus componentes, el lenguaje máquina es el sistema de codificación que tiene cada computadora/procesador para almacenar los programas.
- Software de aplicación, programas que se cargan sobre la memoria para ejecutar una tarea.

Un programa es una lista de instrucciones máquina que al ejecutarse producen un resultado concreto. Deben estar expresados en un lenguaje que entienda la máquina, o bien debemos escribirlos en algún lenguaje que pueda ser traducido a lenguaje máquina utilizando algún tipo de software de base como son los compiladores o los interpretes.

Todo programa comienza con idea, algo que se quiere hacer, generalmente ese algo resulta como solución a un problema específico, la solución de un problema requiere el diseño de un **algoritmo**.

Un "ejemplo" es el algoritmo para llenar un vaso de agua

O de como hacer un sandwich de mantequilla de cacahuete y mermelada de merienda.

Los algoritmos son soluciones abstractas a problemas, que generalmente son codificados en un lenguaje de programación y luego traducidos al lenguaje máquina que es el que una computadora puede ejecutar, para entonces con sus resultados solucionar el problema real para el que se concibieron. Los algoritmos son independientes del lenguaje de programación y de la máquina que lo ejecute, una analogía de la vida real nos la encontramos cuando un cocinero con vista cansada quiere hacer una receta:

La receta de un plato de cocina (algoritmo) puede expresarse en inglés, francés o español (lenguaje) e indicará la misma preparación independientemente del cocinero (máquina). Después el cocinero concreto que lea la receta, debe poder entender lo escrito y como el cocinero es corto de vista necesita sus gafas (compilador) para que le traduzca los garabatos a letras.

Un programa contiene la información codificada del comportamiento deseado o descrito en el algoritmo. Cada modelo de computadora podrá utilizar una forma particular de codificación dependiendo de sus dispositivos físicos, no es lo mismo un móvil que una computadora de sobremesa y que puede cambiar incluso con el modelo de procesador. La forma de codificar programas de una máquina particular se llama código máquina o lenguaje máquina. Los programas en lenguaje máquina son muy difíciles de leer, y se suelen llamar **programas objeto** (obj).

Los lenguajes de programación sirven para representar programas de forma simbólica en forma de texto. Un **lenguaje de programación** es un idioma artificial diseñado para que sea fácilmente entendible por un humano y traducible por una máquina (empleado una pieza de software de base). También son llamados lenguajes de alto nivel.

Se forman con símbolos tomados de un determinado repertorio (componentes **léxicos**). Se construyen siguiendo unas reglas precisas (**sintaxis**). Incorpora unas reglas de **semántica** que determinan el significado de cada construcción correcta.

Aun sin conocer como programar veamos un ejemplo:

```c
#include <stdio.h>

int main(void)
{
    printf("Hola mundo ");
    return 0;
}
```

gcc compila.c -o compila

Tipos de lenguaje de programación

- Lenguajes de Marcado: No son considerados lenguajes de programación como tales se añade códigos en formato texto (marcas) que indican información para la visualización de la información o alguna procesamiento. Ejemplos: markdown, latex, HTML.
- Lenguajes de programación. Es un conjunto de reglas y símbolos que permiten a un programador **escribir instrucciones** que una computadora puede entender y ejecutar. Hay muchos tipos y de muchas categorías

  Se clasifican según su:

  - Nivel de abstracción:
    - Bajo Nivel. Se programa sobre el hardware, ejemplo ensamblador
    - Alto nivel. Tenemos una serie de símbolos y reglas que definen instrucciones mas complejas
  - Estilo de programación
    - Imperativa — Solo veremos esta. Hay que definir el detalle de lo que hay que hacer.
    - Funcional
    - Orientada a objetos
    - Declarativa
    - Guiada por eventos
    - Concurrente
  - Dominio de aplicación
    - Aplicaciones científico tecnológicas
    - Aplicaciones de gestión de la información

  - Aplicaciones de inteligencia Artificial
  - Aplicaciones de programación de sistemas
  - Aplicaciones para la web
- Ámbito de uso
  - Web
  - Móviles
  - Tradicional (equipos fijos y procesamiento local)
  - Hardware (Programación de sistemas, diseño de circuitos,...)
- Forma de traducción
  - Compiladores
  - Intérpretes

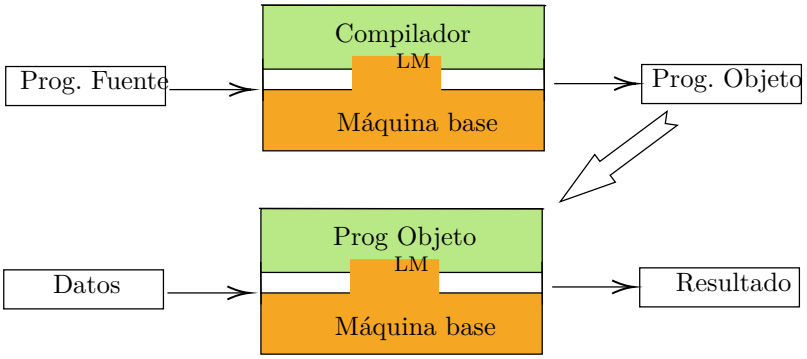
Existen diferentes estrategias para poder ejecutar/conseguir el lenguaje máquina escrito en un lenguaje de programación. Se necesita un software especial que procesará el lenguaje de alto nivel. Estos **procesadores de lenguajes o traductores** manipulan la descripción simbólica de una algoritmo escrito en un lenguaje de programación (programa) para obtener el código objeto que ya si es ejecutable en la máquina. Existen dos grandes categorías en los traductores: Compiladores e interpretes.

Traductores: Compiladores e interpretes

Los **procesadores de lenguajes o traductores** manipulan la descripción simbólica del algoritmo escrito en un lenguaje de programación (programa) para obtener el código objeto que ya si es ejecutable en la máquina. Las dos grandes categorías en los traductores:

- Compilador: una vez traducido el programa fuente, su ejecución (programa objeto) es independiente del compilador (se traduce una vez y se ejecuta múltiples veces).
- Intérprete: el programa fuente se traduce instrucción a instrucción cada vez que se ejecuta (no se crea el programa objeto).

La siguiente figura muestra las diferencia entre ambas estrategias.



Algunos lenguajes de programación

Python

- Fácil de aprender. Su filosofía hace hincapié en una sintaxis que favorezca un código legible.
- Multiparadigma (soporta orientación a objetos, programación imperativa y programación funcional).
- Multiplataforma
- Bibliotecas amplias y variadas
- Interpretado: Los archivos .py requieren tener python disponible para ser ejecutados.

Java

- Lenguaje portable, basado en C.
- Utilizado para aplicaciones en internet.
- A la vez compilado e interpretado. Con el compilador se convierte archivos .java, a un conjunto de instrucciones que recibe el nombre de bytecodes, en un archivo .class. Estas instrucciones son independientes del tipo de ordenador. El intérprete ejecuta estas instrucciones en un ordenador específico (máquina virtual java).

C

- Lenguaje básico de los sistemas UNIX, se inventó para escribir un sistema operativo llamado UNIX.
- Caracterizado por la economía de expresión.
- Utilizado para programación de sistemas.
- Revisiones del lenguaje: C99, C11 y C18.
- Gestión de memoria - Soporta la característica de asignación dinámica de memoria.
- Estándar ISO/IEC 9899:2018 (C18)
- Familia de lenguajes C: C++ y C#.
- Ampliamente utilizado en ingeniería por estar más cerca del dispositivo.
- Los programas escritos en C son eficaces y rápidos.
- C es básicamente una colección de funciones de la biblioteca C; también podemos crear nuestra propia función y añadirla a la biblioteca C.
- C es fácilmente extensible.
- C es un lenguaje robusto con un rico conjunto de funciones y operadores incorporados.
- Compilado. desde el archivo .c se genera un ejecutable .exe. Aunque suele ser trasparente desde los entornos de construcción de programas, la generación del ejecutable tiene dos fases:
    - Compilar: Crea los objetos (.o o .obj).

- Enlazar o Linkar: Une los objetos para crear ejecutables. Busca librerías y las añade si es necesario.

Tendencias en lenguajes de programación

[Evolución de los lenguajes de programación]https://www.youtube.com/watch?v=n6mVXfRCD0c)

El lenguaje de programación C

El lenguaje C fué desarrollado en la segunda mitad de los años 70 por Brian Kernighan y Dennis Ritchie. Es un lenguaje de alto nivel que está estrechamente vinculado con el sistema operativo Unix. No obstante, el lenguaje incluye bastantes facilidades propias de los lenguajes de bajo nivel. El lenguaje C nos permite estructurar los programas mediante acciones y funciones, y además, manipular de forma eficiente contenidos de información elemental como si se utilizase un lenguaje ensamblador.

El precio a pagar por tener un lenguaje orientado a la eficiencia es que su compilador tiene mecanismos de detección de errores más laxos que otros lenguajes de alto nivel. El programador en este lenguaje debe ser muy consciente de la codificación que produce puesto que, la detección de errores puede ser ardua y costosa y que es su responsabilidad llevar el control del programa, por ejemplo controlando no superar los límites de las colecciones de datos (arrays) como veremos más adelante.

El lenguaje C es un buen lenguaje para programar y un mal lenguaje para aprender a programar. No hay que alarmarse por esta afirmación; la realidad es la que es. Todo lenguaje está sujeto a una curva de aprendizaje que hay que superar. Lo importante es habituarse a que no siempre los conceptos teóricos se plasman tal cual en la práctica, y que hay que desarrollar hábitos y emplear metodologías adecuadas para elaborar programas correctos en cualquier lenguaje. En esta asignatura se introducen aspectos básicos del lenguaje C que se irán ampliando paulatinamente en otras asignaturas y probablemente con otros lenguajes de programación.

La resolución de problemas en C suele constar de varias partes: **un entorno** de desarrollo de programas (compilador y editor), **el lenguaje de programación** y **la biblioteca** estándar de C (funciones y/o utilidades preescritas y listas para ser usadas).

Los elementos que definen un **lenguaje de programación** son:

- **Alfabeto**, es decir, los caracteres que pueden usarse
- **Léxico**
  - Palabras y su significado
  - Elementos básicos con los que se componen los programas
- **Sintaxis** Reglas para combinar las palabras, de manera que tengan sentido.
- **Semántica**, significado, ligado a la teoría de la programación estructurada.

Alfabeto de C Símbolos que pueden aparecer en un programa en C

- Letras, exceptuando `ñ` y letras con tilde
- Números y operadores
- Caracteres especiales `( ) & " \` y el espacio
- El carácter `;` es importante en C porque indica el final de una instrucción. ¡¡¡Muy importante!!!

El compilador distingue MAYÚSCULAS y minúsculas.

Léxico de C Todo lenguaje de programación tiene un léxico = elementos básicos con los que se construyen los programas:

- **Palabras clave** o palabras reservadas
  - palabras que tienen un significado especial para el compilador
  - Siempre minúscula: `include` , `define` , `main` , `if` , etc.

En la tabla siguiente se listan todas las palabras clave reservadas por el lenguaje C. Cuando el lenguaje de programación actual es C o C++, estas palabras clave no se pueden abreviar, ni utilizarse como nombres de variable o utilizarse como cualquier otro tipo de identificador, tienen un significado definido en el propio lenguaje.

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed

- **Separadores**
  - espacios en blanco, saltos de línea, tabuladores. Los espacios en blanco permiten identificar las palabras a "compilar". Los saltos de línea y tabuladores en **C** sirven para hacer más legible a los humanos los programas. Solo el carácter `;` es el indicador de fin de línea.
- **Operadores**
  - Representan operaciones como las aritméticas (`+`, `>`), lógicas (`&&`, `||`), de asignación (`=`), etc. También `#`.
- **Literales**
  - Especifican un valor concreto (Números, caracteres y cadenas de caracteres).
- **Identificadores**:

- Nombres de las variables , constantes y funciones definidas por el programador: “perimetro” , “PI”, “calcularRadio” (explicaremos más adelante).
- Las palabras clave no se pueden utilizar como identificadores.

La manera de hacer referencia a los distintos elementos que intervienen en un programa es darles un nombre particular a cada uno. En programación llamamos **identificadores** a los nombres usados para identificar un elemento dado en un programa.

Primer programa en C

Comenzamos con un sencillo programa en C que imprime una línea de texto.

```
1  /*
2  *  Descripción: Programa que saluda  al usuario con el mensaje Hola Mundo
3  *
4  */
5
6  #include <stdio.h>
7
8  // la función main inicia la ejecución del programa
9  int main(void)
10 {
11     printf("Hola mundo ");
12
13 } // fin de la función main
```

Comentarios En cualquier programa los programadores pueden incluir texto que será ignorado por el compilador si es le dice que es un comentario. En C hay dos formas de poner los comentarios.

`/*` para iniciar el comentario que finalizará con `*/`, para los comentarios multilínea. O bien `//` que indica que toda esa línea es un comentario.

En el ejemplo nos encontramos con los dos casos en las líneas de 1 a 3 y la línea 7 y 11.

Por estilo, las primeras líneas de un programa indican en lenguaje entendible por las personas la identificación de ese programa, su autor y la fecha de realización, de forma que habría que extender los comentarios de identificación de este programa de la forma:

```
/*
** Archivo: nombre_archivo.c
** Autor: Fulanito Menganito Blas
** Fecha: 21-02-24
**
** Descripción: Saludo al usuario
**
**
*/
```

Para los ejemplos que se incluyen en este documento y en la asignatura se utilizará otra versión distinta de los comentarios identificativos: se incluirá una breve descripción, el asunto que se cubre en ese código, como `funciones de math.h`, y la práctica o el trabajo individual al que hace referencia.

```
/*
** Descripción: <<texto ilustrativo de la funcionalidad>>
** Asunto: <<tema genérico por ejemplo condicionales, bucles,...>>
** Área: <<práctica o trabajo individual que resuelve>>
*/
```

Directiva del preprocesador `include` La línea

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

es una directiva del preprocesador de C. El preprocesador maneja las líneas que comienzan con `#` antes de la de la compilación.

Esta línea indica al preprocesador que incluya el contenido de la cabecera de entrada/salida estándar estándar (`<stdio.h>`). Este es un archivo que contiene información que el compilador utiliza para asegurarse de que utiliza correctamente las funciones de la librería de entrada/salida estándar como `printf` (línea 10). Esto se verá con detalle más adelante.

Líneas en blanco y espacios en blanco Hemos dejado la línea 8 en blanco. Se utilizan líneas en blanco, caracteres de espacio y caracteres de tabulación para facilitar la lectura de los programas. Juntos, se conocen como espacios en blanco y son generalmente ignorados por el compilador en C en otros lenguajes como Python son parte de su léxico.

Función principal - main La línea

```
int main(void)
{
```

forma parte de todos los programas en C. Los paréntesis después de main indican que `main` es un bloque llamado `function` (función). Los programas en C constan de funciones, una de las cuales debe ser `main` (programa principal).

Todo programa comienza a ejecutarse en la función main. Como buena práctica preceda cada función con un comentario indicando el propósito de la función.

Las funciones pueden devolver información. La palabra clave `int` a la izquierda de `main` indica que main "devuelve" un valor entero (número entero). En temas posteriores se profundizará en este apartado.

Por ahora, basta con incluir la palabra clave `int` a la izquierda de main en cada uno de los programas.

Las funciones también pueden recibir información cuando son llamadas a ejecutarse. El `void` entre paréntesis significa que main no recibe ninguna información.

Una llave izquierda, `{`, comienza el cuerpo de cada función (línea 9). La llave derecha `}`, termina el cuerpo de cada función (línea 11).

Cuando un programa alcanza el cierre de `main`, el programa termina. Las llaves y la parte del programa entre ellas forman un **bloque**. Llamamos **bloque de código** a las instrucciones comprendidas entre dos llaves.

Una instrucción de salida La línea

```
printf("Hola mundo");
```

ordena a la computadora que realice una acción, a saber, mostrar en la pantalla la cadena de caracteres entre comillas. Una cadena se denomina a veces cadena de caracteres, un mensaje o un literal.

Se realiza la "llamada" a la función **printf** para que realice su tarea, el argumento de printf dentro de los paréntesis y el punto y coma (**;**) indica que se ha acabado la instrucción/orden al computador.

Toda instrucción debe terminar con un punto y coma. El "f" en printf significa "formateado".

Cuando se ejecuta, muestra el mensaje *Hola mundo* en la pantalla. Los caracteres normalmente se imprimen como aparecen entre las comillas dobles. Podrían haberse utilizado dos instrucciones:

```
printf("Hola");
printf(" mundo");
```

Recursos adicionales

- Enlace para ver la notación en bits de un número real
- Enlace a Conversor en línea
- Enlace para el estudio de las formas de representación de información numérica
- Videos de Dr. Alberto Prieto sobre la arquitectura de una computadora

Fundamentos de la programación estructurada

Flujo de desarrollo de un programa

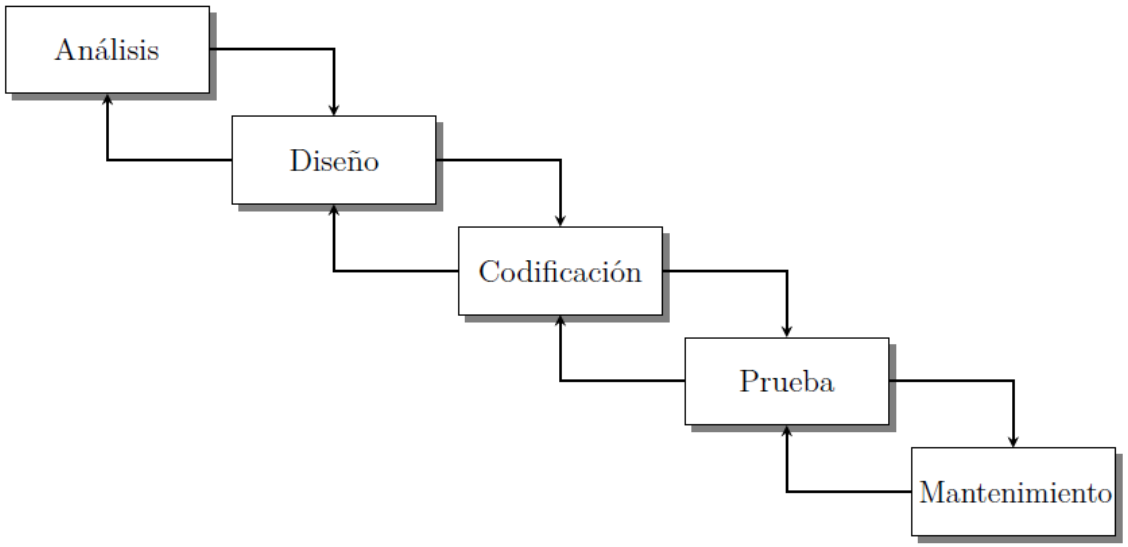
Crear un programa no es diferente de la resolución de problemas en general. Escribir un programa casi es el último paso del proceso de determinar, primero cuál es el problema y después el método que se usará para resolverlo, es decir el algoritmo.

La disciplina de ingeniería que se ocupa de la producción de software se denomina **ingeniería del software**.

La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación, y mantenimiento del software.

El método usado por los profesionales que desarrollan software para entender el problema que se va a solucionar y para crear una solución de software efectiva y apropiada lo llamaremos procedimiento de desarrollo de software.

Este procedimiento, se ilustra en la figura, pero hemos de entender que se trata de una versión simplificada. Buscando un símil en ingeniería de la construcción, no es igual el método para construir un rascacielos que una casa unifamiliar, pero se fundamentan en los mismos principios, los ejemplos que se tratarán en esta asignatura utilizando este mismo símil son "casetas para mascotas".



El **análisis** representa el **Qué**. Conlleva la definición concreta del problema, la especificación de los datos de entrada y salida, la identificación de las tareas a realizar, así como la definición de los criterios de validación indicando cuáles son los casos para los que la solución debe resolver el problema.

Veámoslo sobre un ejemplo: queremos calcular el **área de una figura geométrica**. En el análisis estudiamos cuál es el problema exactamente y para que casos será válido el programa a construir, por ejemplo nuestras figuras serán triángulos y la información de la que disponemos es las coordenadas relativas de cada vértice del triángulo. Los valores de entrada son por tanto tres parejas de números y el resultado será un número real con la superficie del triángulo.

El **diseño** supone la elaboración de la solución, es decir, el **Cómo**. Conlleva la definición completa de los algoritmos y datos que se van a utilizar. Habitualmente se comienza con un diseño global o arquitectónico, donde se estudia la estructura y pasos a seguir sin utilizar ningún tipo de notación (incluso lenguaje natural - *preAlgoritmo*) para refinar el resultado mediante el **diseño detallado** de los algoritmos, utilizando una notación propia de la programación estructurada como puede ser el pseudocódigo o los diagramas de flujo de datos. Se trata de un proceso creativo que se dan en dos pasos de refinamiento, primero se hace el **diseño preliminar** donde se define la estructura de nuestro problema y la estrategia de resolución. Después durante el **diseño detallado** se define los pasos detallados que recogen el algoritmo.

Para el ejemplo del área del triángulo, el diseño arquitectónico sería decidir si se utiliza la formula clásica de dividir por dos el resultado de base por altura, o si bien la formula de Heron a partir del semiperímetro, que es la elegida en este caso. No obstante en ambos casos habrá que comprobar que es un triangulo propio (a < b+c).

El diseño detallado consiste en identificar los pasos correctos.

- Guardar coordenadas de tres vértices: v1, v2, v3.
- Calcular distancia a de v1 a v2.
- Calcular distancia b de v2 a v3.
- Calcular distancia c de v3 a v2.
- Buscar el lado más largo.
- Comprobar que su valor es más pequeño que la suma de los otros dos.
- Si se cumple la condición de Heron calcular el semiperímetro (a+b+c)/2
- area = raiz cuadrada (p(p-a)(p-b)(p-c)), siendo p el semiperímetro.

La **codificación** es la traducción a un lenguaje de programación de los algoritmos. Traducción del diseño a lenguaje de programación de alto nivel (programa fuente) que se traduce a lenguaje máquina (programa ejecutable), utilizando un compilador o un intérprete dependiendo del lenguaje.

La **prueba** o validación consiste en la ejecución del código escrito para comprobar que las salidas generadas se corresponden con la solución al problema según las entradas definidas para cada caso. Estos casos de prueba se construyen durante el análisis y representan las situaciones a las que se puede dar solución aplicando el algoritmo diseñado y codificado en el lenguaje de programación escogido.

El **mantenimiento** cambios en el programa debidos a:

- Corrección de nuevos errores.
- Adaptación a nuevos entornos (sistemas operativos, periféricos,...).
- Ampliaciones funcionales o de rendimiento.

Programación es la colección de actividades que rodean la descripción, el desarrollo y la implementación efectiva de soluciones algorítmicas a problemas bien especificados. En el ámbito de esta asignatura sobre el ciclo clásico de la ingeniería del software dejaríamos fuera el mantenimiento y habitualmente el trabajo de análisis esta simplificado puesto que se tratará de problemas bastante acotados y de complejidad mediana o pequeña. Con respecto a la fase de diseño, se hará enfocándose en la implementación final en lenguaje C.

Programación no es "codificar en un lenguaje de programación particular o para una arquitectura máquina particular", sino que consisten en el dominio de técnicas altamente estilizadas en un lenguaje de programación particular.

No obstante se deben cumplir unos criterios de calidad que definen lo que debe cumplir un programa:

- **Corrección**: la entrada definida produce los resultados requeridos (el programa se ajusta a la especificación).
- **Claridad**: prácticamente todos los programas se modificarán en el futuro. Su contenido debe ser entendible por personas distintas a su autor o por el mismo autor pasado un tiempo.
- **Eficiencia**: consumo óptimo de recursos de computación (tiempo de CPU, memoria,...).

A estos tres criterios básicos podemos añadir

- **Amigabilidad**: facilidad de uso para el usuario. En ciertos casos puede producir ineficiencia y se ha de priorizar.
- **Robustez**: entradas no definidas - mensajes de aviso sin generar bloqueo. Este criterio se relaja en ciertas condiciones.

Fases de construcción de un programa en C

La implementación de programas C suele pasar por seis fases para llegar a poder ser ejecutados: editar, preprocesar, compilar, enlazar, cargar y ejecutar, pero cuidado antes se ha de haber pasado por las etapas de análisis y diseño. Aunque en los entornos actuales esta división puede ser trasparente, es decir el usuario no ve de forma separada las fases y puede llegar a trasladarse a pulsar un botón en el entorno de desarrollo.

Fase 1: Creación de un programa La fase 1 consiste en editar un fichero en un programa editor: Los entornos de desarrollo de desarrollo C y C++ (IDEs) como Microsoft Visual Studio y Apple Xcode tienen editores integrados. Se escribe un programa C en el editor, se hacen correcciones si es necesario, luego se almacena el programa en un dispositivo de almacenamiento secundario como un disco duro, siendo necesario "guardar" un archivo. En C los archivos deben terminar con la extensión **.c**. También es posible trabajar sobre almacenamientos remotos o "codespaces" espacios de codificación donde el almacenamiento se hace remoto sobre herramientas tipo Git, como GitHub. Como recomendación si es posible se recomienda utilizar la opción de "autoguardado".

Fases 2 y 3: Preprocesamiento y compilación de un programa C Se da la orden para compilar el programa, entonces el compilador traduce el programa C a código de lenguaje de máquina (también denominado como código objeto). En un sistema C, el comando de compilación invoca un programa preprocesador antes de que comience la fase de traducción del compilador.

El preprocesador C obedece comandos especiales llamados directivas del preprocesador, que realizan manipulaciones de texto en los archivos de código fuente de un programa. Estas manipulaciones consisten en insertar el contenido de otros archivos y varios reemplazos de texto.

En la **Fase 3** el compilador traduce el programa C a código en lenguaje máquina:

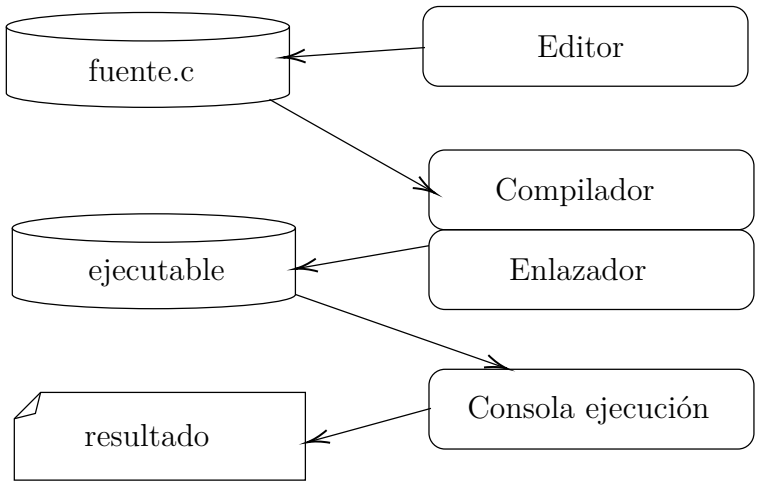
Un error de sintaxis ocurre cuando el compilador no puede reconocer una instrucción porque viola las reglas del lenguaje. El compilador emite un mensaje de error para ayudarle a localizar y corregir la instrucción incorrecta. El estándar C no especifica la redacción de los mensajes de error emitidos por el compilador, por lo que los mensajes pueden diferir entre los distintos sistemas. Los errores de sintaxis también se denominan errores de compilación o errores en tiempo de compilación.

Fase 4: de Enlazado o Linker Los programas en C suelen utilizar funciones definidas en otros lugares, como en las bibliotecas estándar bibliotecas de código abierto o bibliotecas privadas de un proyecto concreto. El código objeto producido por el compilador de C suele contener "agujeros" debidos a estas partes que faltan. A enlazador enlaza el código objeto de un programa con el código de las funciones que faltan para producir una imagen ejecutable (sin las piezas que faltan).

Fase 5 y 6: Carga o lectura y ejecución Antes de que un programa pueda ejecutarse, el sistema operativo debe cargarlo en memoria. El cargador toma la imagen ejecutable del disco y la transfiere a la memoria. Además, también se cargan los componentes adicionales de las bibliotecas compartidas que soportan el programa.

Finalmente, en la última fase, el ordenador, bajo el control de su CPU, ejecuta el programa instrucción a instrucción.

En los entornos actuales de codificación bastará con pulsar un único botón una vez editado el archivo y se lanzan todas las fases anteriores.



Veamos un ejemplo:

- Editamos con un editor local bloc de notas generando un archivo .c
- Subimos al codespace simplemente arrastrando el archivo `gcc hola.c -o hola.exe`
- Ejecutamos en la consola `./hola.exe`

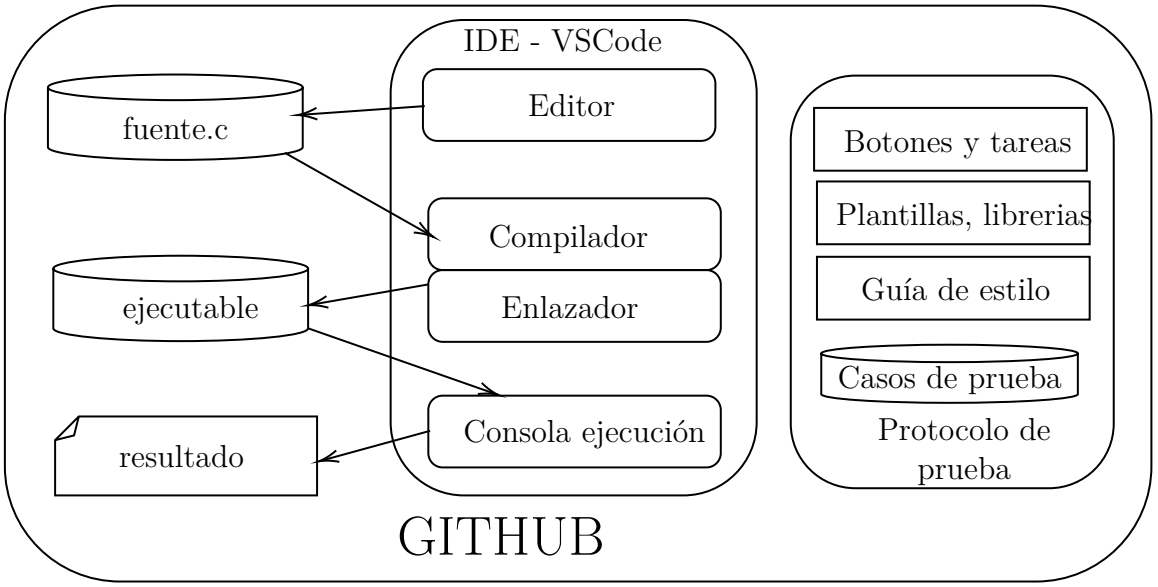
Ahora vamos a un entorno totalmente online - https://www.onlinegdb.com/online_c_compiler

O bien lo hacemos sobre VSCode en el codespace de Github.

Entorno de trabajo en la asignatura de programación

No obstante para trabajar más cómodamente se utilizan extensiones en VSCode que facilitan el trabajo, que validan parte del léxico de C o que facilitan la indentación del código.

Además hemos definido un entorno completo de ejecución, prueba y evaluación para la asignatura que presenta la siguiente estructura, pero sobre la que se irá profundizando poco a poco.



Pero no olvidemos: "¡Qué los árboles no nos impidan ver el bosque!". Las herramientas son lo de menos. Lo que nos interesa es aprender a programar. Y programar es diseñar y después escribir lo diseñado en un lenguaje de programación, como se compile o ejecute es lo de menos.

Tipos de datos simples

En este apartado se presentan los elementos mínimos de un lenguaje de **programación imperativo**. Este conjunto de elementos se particulariza para el lenguaje C.

Identificadores

En programación llamamos **identificadores** a los nombres usados para representar y utilizar un elemento dado en un programa (variables, constantes con nombre, o funciones...). En el ejemplo del "Hola mundo" "printf" es el identificador que los desarrolladores de C en la librería "stdio.h" le pusieron a la instrucción que permite mostrar algo en la pantalla. Este identificador no forma parte de las palabras reservadas de C, lo crearon quienes dieron forma a la librería "stdio.h". Aunque en la práctica pueda parecer parte del lenguaje C no lo es, sin incluir la librería el compilador no lo entiende.

Reglas generales para los identificadores:

- Utilizar solo letras, números y subrayados ('_').
- Sin espacios en blanco intermedios. El lenguaje utiliza los espacios para "entender" y compilar el código, al igual que el **;** da fin a la instrucción.
- El primer carácter debe ser una letra.
- No incluir la letra **'ñ'** ni las vocales acentuadas.
- No utilizar caracteres especiales.
- No usar palabras reservadas del lenguaje.
- Se distingue entre mayúsculas y minúsculas. **¡¡importante!!**

Recomendaciones:

- No utilizar identificadores que ya aparecen en las librerías habituales "printf", "sqrt".
- Utilizar siempre el mismo criterio en cuanto a la combinación de mayúsculas y minúsculas.
- Utilizar nombres significativos.
- Recordar que los acentos no son parte de los caracteres permitidos, ni la ñ.

Lo habitual es que los equipos de programadores definan una guía de estilo para dar una forma unificada a los programas que construyen donde se fija también como nombrar los identificadores. Existen diversas notaciones unificadas de formas de nombrar identificadores, al final de este tema se incluyen las normas a seguir en esta asignatura.

Clasificación de los tipos de datos

Recordemos que un computador es un máquina de tratamiento de información que manipula distintos tipos de datos. Un dato es elemento de información que es objeto de operación por parte de la computadora que tiene un tipo. Un tipo de dato es la propiedad abstracta asociada a la representación del dato en la computadora. Se caracteriza por:

- Implementación: representación interna.
- Dominio: rango de valores permitidos.
- Operaciones permitidas.

No obstante la información además de clasificarse por tu tipo, se distingue entre si puede o cambiar su valor entre varios posibles, en esta clasificación tenemos los **datos** y las **constantes**. Esta distinción es importante de cara a la representación sobre la máquina donde se ejecuten los algoritmos, los datos también se llaman **variables**, concepto retomaremos con detalle más adelante.

Clasificación de los tipos de datos:

- **Simples estándar** Simples o básicos (no estructurados):
    - Numéricos:
        - Entero

    - Real

  - Lógico o booleano
  - Carácter

- **Simples no estándar** o tipos enumerados. Por ejemplo las notas cualitativas de un estudiante.`{No presentado, Suspenso, Aprobado, Notable, Sobresaliente, Matricula}`
- **Compuestos o estructurados** o derivados:

  - Texto (cadena de caracteres)
  - Otros, incluye los definidos por el usuario (se estudiarán más adelante)

Las particularidades (implementación, dominio y operaciones) de cada tipo pueden variar entre lenguajes de programación. Pero se verán primero de forma general, para después particularizar su comportamiento en C.

Tipo numérico entero

La información se representa en los computadores utilizando un conjunto de celdas de memoria que almacenan dígitos binarios (0 o 1). La cantidad y el rango de los números enteros y reales en matemáticas es un conjunto infinito. El ordenador, por contra, dispone de un número finito de celdas en las que almacenar bits. La principal propiedad de un entero es que permite una **representación exacta**.

En lenguajes como C, de forma nativa, el tamaño máximo de un valor entero, tipo `int`, está determinado por la arquitectura del ordenador concreto que se utilice, típicamente 32 celdas binarias, bits, o sea 4 bytes (8 bits son 1 byte). Sin embargo, en Python, un lenguaje de más alto nivel, no hay un límite predeterminado para el valor absoluto del mayor entero representable. Este valor puede crecer ocupando más y más celdas hasta alcanzar los límites de la memoria disponible en el ordenador, a costa de la eficiencia.

- Implementación:

  - 2 bytes/4 bytes, complemento a 2

- Dominio

  - [-32768, 32767] (2 bytes) entero (**int**)
  - [-2147483648, 2147483647] (4 bytes) entero largo (**long**)

- Operaciones

  `+` Suma de enteros

  `-` Resta de enteros

  `*` Multiplicación de enteros

`/` DIV División de enteros (cociente)

Modulo `%` Resto de la división

`+` (unario) Identidad de enteros

`-` (unario) Cambio de signo

Tipo numérico real

Para los números reales, se utiliza una representación muy diferente a la de los enteros, denominada de punto (o coma) flotante, definida en el estándar IEEE 754. El tipo de dato se denomina **float** por ese motivo. El mayor y menor número real representable en valor absoluto, está en el rango aproximado [1e-323,1.7e308]. Esta es la consecuencia de utilizar un total de 64 bits, 8 bytes, para esta representación.

Se debe ser consciente del carácter limitado de la representación de los float. No solo no se puede expresar un número real tan grande como se quiera, sino que la precisión no es infinita y, por tanto, relativamente pocos números reales se pueden representar exactamente. Los cercanos a cero también tienen problemas.

Su aritmética es **aproximada** aunque permite un rango de valores más amplio.

- Implementación
  - IEEE-754, 4/8 bytes (simple/doble precisión)
- Dominio
  - -3.4E+38..-1.2E-38, 0, +1.2E-38..+3.4E+38 (**float**)
  - -1.7E+308..-2.3E-308, 0, +2.3E-308..+1.7E+308 (**double**)
- Operaciones

  `+` Suma de reales

  `-` Resta de reales

  `*` Multiplicación de reales

  `/` División de reales

  `+` (unario) Identidad de reales

  `-` (unario) Cambio de signo

Tipo lógico

Recoge la verdad o no de una situación. Este tipo de datos **no** está disponible en C. Debemos utilizar 2 bytes y emplear un entero para representar esta misma idea.

- Implementación -1 byte
- Dominio
  - Verdadero (true, 1, <>0)
  - Falso (false, 0)
- Operaciones `no` negación

  `y` conjunción

  `o` disyunción

Tipo de dato carácter

- Implementación:
  - 1 byte/ 2 bytes (**char**)
- Dominio:
  - Juego de caracteres disponibles en la computadora (ASCII, Unicode)
- Operaciones: Funciones que se verán más adelante.

Tipo de datos simples no estándar

Se definen mediante especificación con identificadores de los valores de su dominio (**enumeración**) ó mediante un subconjunto de un tipo ordinal (subrango). Son tipos de datos internos al programa no se pueden involucrar en operaciones de E/S y son isomorfos a un subconjunto de los enteros.

```
int entero = 10;
float decimal = 3.14;
double mayorPrecision = 3.1415926535;
char caracter = 'A';

enum Meses {ENERO, FEBRERO, MARZO, ABRIL, MAYO, JUNIO, JULIO,
            AGOSTO, SEPTIEMBRE, OCTUBRE, NOVIEMBRE, DICIEMBRE};
enum Meses miMes = ABRIL;
enum {FALSO = 0,    VERDADERO = 1}
```

Tipo texto

Los computadores son conocidos por su capacidad para trabajar con valores numéricos. De igual forma, cualquier usuario sabe que frecuentemente demandan otros tipos de datos, de carácter alfanumérico.

El manejo de textos es una tarea recurrente en las aplicaciones de los ordenadores (procesadores de texto, correos electrónicos, sistemas de mensajería, etc.). En todos estos casos, las cadenas de caracteres juegan un papel fundamental.

Una cadena de caracteres, tipo de dato **string** (no disponible como tal en C), representa precisamente eso: una ristra de caracteres alfanuméricos que puede ser tratada como una unidad.

En C se distingue entre caracteres (comillas simples 'a') y cadenas de caracteres (comillas dobles, "cadena").

Variables, operadores y expresiones

Constantes

Son valores que no cambian durante la ejecución del programa. Hay dos tipos de constantes (literales y simbólicas):

Constante literal Cualquier valor escrito directamente en una instrucción de un programa es una constante literal que puede ser de distintos tipos. Ej: 3.14159

Constantes Enteras: 10, -3987. Si no caben en un entero se almacenan en forma entero largo.

- Definición de constantes grandes directamente:
  - 123L, 123l
- Definición de constantes sin signo: - 45U, 45u
- Definición de constantes grandes sin signo:
  - 234UL (U antes de L)
- Enteros no decimales:
  - octales: 0123 (0 delante)
  - hexadecimales: 0X478 ó 0x478 (0X ó 0x delante).

Constantes Reales: punto fijo: 146.09, -234.3

Notación científica: 2.23E-15

Las constantes reales se almacenan en forma double.

Constantes de Caracteres: entre apóstrofes: 'A'. - en octal ASCII: '\101'.

Secuencias de escape (códigos de representación inconfundible: independientes del sistema de codificación). Permiten enviar caracteres de control no gráficos a un dispositivo de pantalla. Todas ellas comienzan con una barra invertida (\) seguida de otro carácter. En tiempo de ejecución, las secuencias de escape se sustituyen por los caracteres adecuados

`'\n'`	nueva línea
`'\t'`	tabulador horizontal
`'\v'`	tabulador vertical
`'\b'`	retroceso
`'\f'`	avance de página
`'\r'`	retorno de carro
`'\a'`	sonido
`'\0'`	nulo (marca de fin de cadena)
`'\?'`	signo de interrogación
`'\\'`	\ (barra invertida)
`'\''`	' (apóstrofe)
`'\"'`	" (comillas)

Constante simbólica (con nombre) Es un valor de dato constante que se referencia mediante un nombre (identificador).

```
PI
```

En C es posible definir constantes simbólicas utilizando dos alternativas, con la palabra reservada `const` o con la directiva del precompilador `#define`

```
const int NOMBRE = 10;
#define NOMBRECONSTANTE valor
```

La diferencia entre el uso de `const` y el uso de `#define` está en que mediante el primero se declara una constante que tiene un tratamiento semejante a una variable, pero sin poder cambiar su valor mientras que mediante define se indica que escribir el nombre especificado equivale a escribir el valor (de hecho se sustituye en pre-compilación por él), con una correspondencia directa.

Por ejemplo:

```
const int JUGADORES = 5;
#define JUGADORES 5
```

En la primera se indica que JUGADORES es una constante de tipo `int` (entero) mientras que en la segunda se indica que donde aparezca en el código la palabra JUGADORES deberá ser reemplazada por 5 directamente. En general, usar `#define` supone que la compilación sea más rápida. Por ello su uso resultará recomendable cuando existan ciertos valores numéricos que tengan un significado especial, valor constante y uso frecuente dentro del código. Las constantes definidas con **`#define`** se denominan constantes simbólicas, y algunas de ellas existen de forma predeterminada en el lenguaje.

A continuación se incluye un ejemplo de uso de secuencias de escape y constantes simbólicas. La combinación `%s` permite formatear la salida, siendo un parámetro de la función `printf`.

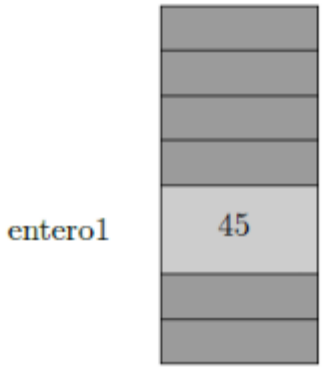
```
#define NOMBRE "Computadora"

// la función main inicia la ejecución del programa
int main(void)
{
    printf("\n \n Hola mundo, me llamo  %s \n \n", NOMBRE);
    return 0;
} // fin de la función main
```

Variables

Se ha de distinguir entre dos conceptos, el de variable y el de identificador de la variable. Poniendo un ejemplo en un contexto distinto: Una cosa es la persona y otra el nombre que se utiliza para referenciarla. Isabel María del Águila es una persona, que se identifica como "profesora" o bien como "mamá", siendo la misma persona.

Variable es zona de memoria central que se referencia mediante un nombre o identificador, en lugar de por su dirección, donde se puede almacenar el valor de un dato que puede cambiar durante la ejecución del programa.




```
int main(void)
{

    int entero1 = 0; // guarda el primer numero introducido por el usuario
    int entero2 = 0; // guarda el segundo numero introducido por el usuario

    printf("Dame el primer entero: "); // mensaje en pantalla
    scanf("%d", &entero1);             // lee un entero

    printf("Dame el segundo entero: "); // mensaje en pantalla
    scanf("%d", &entero2);              // lee un entero

    int suma = 0;             // variable donde se guardará la suma
    suma = entero1 + entero2; // asigna el resultado de la operación a suma

    printf("La suma es %d\n", suma); // muestra el resultado

    return 0;
} // fin de main
```

Definición de variables Las líneas

```
int entero1 = 0; // contendrá el primer número que introduzca el usuario
int entero2 = 0; // contendrá el segundo número que introduzca el usuario
```

son definiciones. Los nombres `entero1` y `entero2` son variables-ubicaciones en memoria donde el programa puede almacenar valores para su uso posterior.

Estas definiciones indican que **entero1** y **entero2** son de tipo **int**. Esto significa que contendrán números enteros. enteros, como 7, -11, 0 y 31914 y que se reservará en memoria espacio (segun el tipo) para poderlos almacenar.

También se inicializan cada variable a 0 siguiendo el nombre de la variable con un `=` y un valor. Aunque no es necesario inicializar explícitamente cada variable, hacerlo ayudará a evitar muchos problemas si se usan sin haberlas inicializado.

Es necesario **definir** las variables antes de utilizarlas, a esto se le llama **declarar** una variable. Todas las variables deben definirse con un nombre o identificador y un tipo antes de que puedan utilizarse en un programa. Se puede colocar cada definición de variable en cualquier lugar de `main` antes de que la variable de esa variable en el código. En general, debe definir las variables cerca de su primer uso o bien al principio de la función que la utiliza.

Tal como hemos visto en el ejemplo, en el lenguaje C es necesario indicar el tipo que va a soportar una variable, para que así en tiempo de compilación se reserven en memoria las posiciones necesarias para manejarlo adecuadamente.

Un nombre de variable puede ser cualquier identificador válido. Cada identificador puede estar formado por letras, dígitos y guiones bajos (_), pero no puede empezar por un dígito. C distingue entre mayúsculas y minúsculas, por lo que `a1` y `A1` son identificadores diferentes.

Elegir nombres de variables significativos ayuda a que un programa esté autodocumentado, por lo que se necesitan menos comentarios.

Evite comenzar los identificadores con un guión bajo (_) para evitar conflictos con los identificadores generados por el compilador y los identificadores de la librerías estándar.

La función scanf y las entradas con formato Las dos linea mostradas a continuación sirven para que el usuario pueda introducir por teclado el valor de la variable con la que trabajar y se tratarán con detalle en el tema siguiente.

```
printf("Dame el primer entero: "); // mensaje en pantalla
scanf("%d", &entero1); // lee un entero
```

La primera sirve solo para mostrar un mensaje que ayude al usuario a dar un valor (`printf`). Se utiliza `scanf` para obtener un valor del usuario. La función lee de la entrada estándar, que normalmente es el teclado. La “f” en scanf significa “formateado”. Este scanf tiene dos argumentos- `%d` y `&entero1`. El primero es la cadena de control de formato. Indica el tipo de datos que el usuario debe introducir.

`%d` especifica que los datos deben ser un número entero. La `d` significa “entero decimal”. Un carácter `%` indica que especificación de conversión de formato.

El segundo argumento de `scanf` comienza con un ampersand (&) seguido del nombre de la variable. El `&` es el operador de dirección y, cuando se combina con el nombre de la variable, indica a scanf la ubicación (o dirección) en memoria de la variable entero1. scanf entonces almacena el valor que el usuario introduce en esa posición de memoria.

El uso del ampersand (`&`) es a menudo confuso para los programadores principiantes y para la gente que ha programado en otros lenguajes que no utilizan el ampersand, dirección. Por ahora bastará con preceder cada variable en cada llamada a scanf con un ampersand. Algunas variaciones a esta regla se tratan más adelante.

Solicitud e introducción del segundo número entero

```
printf("Dame el segundo entero: "); // mensaje en pantalla
scanf("%d", &entero2); // lee un entero
```

obtiene del usuario un valor para la variable **entero2**.

Declaración de la variable sum

```
int suma = 0; // variable donde se guardará la suma
```

define la variable `int sum` y la inicializa a `0` antes de que utilicemos `sum` en la línea siguiente.

Instrucción de Asignación

```
suma = entero1 + entero2; // asigna el resultado de la operación a suma
```

Calcula el total de la suma de las variables, luego asigna el resultado a la variable **suma** utilizando el operador de asignación (=).

Una vez que tenemos valores para `entero1` y `entero2`, la línea

```
suma = entero1 + entero2; // asigna el total a suma
```

suma estos valores y coloca el total en la variable suma, reemplazando su valor anterior. Aquí se combina una operación (+) con una operación de asignación (=).

Como se intercambian el contenido de dos variables en C:

```
// Intercambia el contenido de a y b utilizando una variable temporal
temp = a;
a = b;
```

Ciclo de vida de una variable Como resumen diremos que a una variable se le aplican cuatro acciones que definen los estados por los que pasa la variable y que no todos son necesarios dependiendo del tipo de variable o el lenguaje de programación que se utilice

- **Declaración**: Se indica el identificador asociado a la variable y su tipo. Es obligatorio en C, pero en python o en R no.
- **Inicialización**: Asignación de valor inicial a la variable. Consiste un dar un valor inicial fijo que prevenga de los problemas que podrían aparecer si se utiliza la variable antes de asignarle un valor. No es estrictamente necesaria, pero es muy recomendable y se puede incluir en la declaración

```
int i=0;
```

- **Utilización**: Durante la ejecución del programa una variable puede se utilizar en las instrucciones de dos formas básicas: dentro de expresiones o bien en operaciones de asignación. No se debe utilizar la variable en una expresión si no ha sido asignada o inicializada primero.
- **Destrucción**: Liberar la memoria asignada a esa variable. Esta operación es especialmente relevante cuando se utilizan estructuras de datos dinámicas referenciadas por punteros. En los ejemplos de esta asignatura no será necesario realizar destrucciones de variables. El programa recoge la basura, es decir libera la memoria que no se referenciada por un identificador.

Expresiones

Una **Expresión** es una combinación de constantes, variables, símbolos de operación y paréntesis que da como resultado un valor (de un tipo de datos determinado) también se le llama **operación**. Su formato es el clásico de las matemáticas.

Ejemplos:

$5/2 + 4 * 4$

$(x + y)/2$

$2 * \text{PI} * \text{radio}$

Elementos de una expresión:

- Operandos: valores constantes o variables.
- Operadores: manipuladores de los datos.

Los operandos de un operador deben de ser de un tipo de datos específico. Una operación/expresión genera un resultado de un tipo concreto.

Tipos de datos en C y como se declaran

Tipo	C	Rango	Tamaño
Enteros	short int	-32768 a 32767	2 bytes
	signed short int	-32768 a 32767	2 bytes
	unsigned short int	0 a 65535	2 bytes
	int	-2147483648 a 2147483647	4 bytes
	signed int	-2147483648 a 2147483647	4 bytes
	long int	-2147483648 a 2147483647	4 bytes
	signed long int	-2147483648 a 2147483647	4 bytes
	unsigned int	0 a 4294967295	4 bytes
	unsigned long int	0 a 4294967295	4 bytes
	long long int	-9223372036854775808 a 9223372036854775807	8 bytes
	unsigned long long int	0 a 18,446,744,073,709,551,615	8 bytes
Reales	float	$1{,}4 * 10^{-45}$ a $3{,}4 * 10^{38}$	4 byes
	double	$5{,}0 * 10^{-324}$ a $1{,}8 * 10^{308}$	8 bytes
Lógicos	int	0 (falso)	1 byte
	char	0 (falso)	1 byte
Carácter	char (tabla ASCII)		1 byte
	signed char	-128..127	1 byte
	unsigned char	0..255	1 byte
Cadenas	char [n]		n bytes

`signed` es el valor por defecto y no es necesario utilizarlo. Además, para los enteros se puede obviar `int`. Por tanto para definir un entero largo con signo se puede utilizar `signed long int` o solo `long`.

En el siguiente programa se muestran los límites y tamaños:

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

int main()
{
    printf("*\n");
    printf("Carácter con signo\n");
    printf("Mínimo: %d\n",CHAR_MIN);
    printf("Máximo: %d\n",CHAR_MAX);
    printf("*\n");
    printf("Carácter sin signo\n");
    printf("Mínimo: 0\n");
    printf("Máximo: %u\n",UCHAR_MAX);
    printf("*\n");

// Tamaños de tipos de datos primitivos
    printf("Tamaño de char: %zu bytes\n", sizeof(char));
    printf("Tamaño de short: %zu bytes\n", sizeof(short));
    printf("Tamaño de int: %zu bytes\n", sizeof(int));
    printf("Tamaño de long: %zu bytes\n", sizeof(long));
    printf("Tamaño de float: %zu bytes\n", sizeof(float));
    printf("Tamaño de double: %zu bytes\n", sizeof(double));
    printf("Tamaño de long long: %zu bytes\n", sizeof(long long));

    return 0;
}
```

Operadores en C Los operadores permiten definir operaciones entre variables y/o constantes. Existen muchos tipos de operadores distintos en función del tipo de resultado cuando se aplica o del número de constantes o variables necesarias para su utilización, es decir su modo de funcionamiento.

Según el modo de funcionamiento de los operadores pueden ser:

- Operadores unarios: solo se aplican a un operando

  `-1`

- Operadores binarios: Se aplican a dos operandos

  `1 + entero1`

Según tipo de operandos y de salida (resultado) obtenida los operadores pueden ser:

Tipo de operador	Tipo de operandos	Tipo de salida
Aritmético	numérico	numérico
Relacional	numérico/carácter/texto	lógico
Lógico	lógico	lógico
Carácter/texto	carácter/texto	carácter/texto
A nivel de bits	numérico/carácter	numérico/carácter
Condicional	lógicos	instrucción
Asignación	identificador/ expresión	-

- Aritméticos:
  - Binarios: `+` `-` `*` `/` `%` (resto)
  - Unarios: `-` (cambio de signo)
- Relacionales y lógicos:
  - Relacionales: `>=` `<` `<=`
  - De igualdad: `==` (igual a) `!=` (distinto de)
  - Conectivas lógicas: `&&` (“and” lógico) `||` (“or” lógico)
  - Negación lógica: `!` (“not” lógico)

p	q	p && q	p || q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

En C no hay tipos lógicos. En su lugar se utilizan valores numéricos: verdadero<>0 y falso=0.

- Incremento y decremento:

- Incrementa en uno una variable. ++
- Decrementa en uno una variable. --

Si se utilizan como prefijo (delante de la variable) producen el efecto antes de utilizar la variable en la expresión en que esté. Si se utilizan como sufijo, producen su efecto después de utilizar la variable en la expresión en que esté. Su utilización genera un código más eficiente.

Nota: no usarlas en variables que se empleen más de una vez en una expresión ó como argumentos de una función.

- Lógicos para bits:
  - Binarios: &Y lógico a nivel de bits (bit a bit).
    | O lógico a nivel de bits.
    ^ O lógico exclusivo a nivel de bits.
    << Rotación a la izq. (`x<<y  x*2y`)
    >> Rotación a la der. (`x<<y  x/2y`)
  - Unario: ~ Complemento a 1.

Los operadores a nivel de bit en C ofrecen ventajas principalmente en situaciones donde se necesita realizar manipulaciones directas sobre los bits de una variable, como en la programación de sistemas embebidos, manipulación de datos comprimidos, implementación de algoritmos de cifrado.

```
*/

#include <stdio.h>

int main() {
```

- Operador condicional:

```
expresión1 ? expresión2 : expresión3
```

Este operador (**?:**) evalúa la **expresión1**, si esta es verdadera, da el resultado de **expresión2**, y en caso contrario da el resultado de **expresión3**. Ejemplos:

```
 c=(a>b)?a:b;

printf("%s\n", calificacion >=70 ? "Aprobado" : "Suspenso");
```

- Operador de asignación

  =

Se considera como un operador con la mínima prioridad. Es decir lo último que se hace es la asignación.

Funciones internas Representan operaciones no elementales que se incorporan en las implementaciones de los lenguajes de programación equivalen a unos operadores especiales que tienen aspecto de funciones matemáticas. Ej:

```
#include <math.h>
```

Función	tipo	significado
acos(d)	double	Arco coseno (0-pi) del argumento (-1,+1)
asin(d)	double	Arco seno (-pi/2,+pi/2) del argumento (-1,+1)
atan(d)	double	Arco tangente (-pi/2,+pi/2) del argumento
atan2(d1,d2)	double	Devuelve el arco tangente de d1/d2
ceil(d)	double	Devuelve el entero mas pequeño mayor ó igual al argumento (redondeo hacia arriba).
cos(d)	double	Coseno del argumento expresado en radianes
cosh(d)	double	Coseno hiperbólico del argumento
exp(d)	double	Exponencial del argumento
fabs(d)	double	Valor absoluto de un número real
floor(d)	double	Redondeo hacia abajo (mayor entero menor ó igual al argumento)
fmod(d1,d2)	double	Devuelve el resto de d1/d2 con el mismo signo que d1
labs(l)	long int	Valor absoluto de un entero largo
log(d)	double	Logaritmo natural del argumento
log10(d)	double	Logaritmo decimal del argumento
pow(d1,d2)	double	Calcula d1 elevado a d2
sin(d)	double	Seno del argumento expresado en radianes
sinh(d)	double	Seno hiperbólico del argumento
sqrt(d)	double	Raíz cuadrada positiva del argumento
tan(d)	double	Tangente del argumento expresado en radianes
tanh(d)	double	Tangente hiperbólica del argumento

Nota: la segunda columna indica el tipo de dato devuelto por la función. En la primera columna, los argumentos que aparecen son: c: carácter, d: doble precisión, s: cadena.

```
#include <stdlib.h>
#include <math.h>

int main()
{
   system("cls||clear");
   double x1, y1, x2, y2;

   printf("Ingrese las coordenadas del primer punto (x1 y1): ");
   scanf("%lf %lf", &x1, &y1);
   printf("Ingrese las coordenadas del segundo punto (x2 y2): ");
   scanf("%lf %lf", &x2, &y2);

   double distancia = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

   printf("La distancia euclidiana entre los dos puntos es: %.2lf\n", distancia);

   return 0;
}
```

Reglas de prioridad Precedencia y orden de evaluación:

Operador	Asociatividad
( ) [ ] -> .	I-D (Izquierda-Derecha)
! ~ ++ – - (tipo) * & sizeof	D-I
* / %	I-D
+ -	I-D
« »	I-D
< <= > >=	I-D
== !=	I-D
&	I-D
|	I-D
&&	I-D
||	I-D
?:	D-I
= += -= ...	D-I
,	I-D

Nota: hay que tener cuidado con la utilización de operandos dependientes del resultado de otros operandos.

Normas de estilo para la programación en lenguaje C. Variables

Un mismo programa se puede escribir de varias formas. Todas ellas pueden ser compiladas correctamente y obtener un ejecutable, pero algunas de ellas son más fáciles de entender por las personas que otras.

> "Cualquier tonto puede escribir código que una computadora pueda entender.
> Los buenos programadores escriben código que los humanos pueden entender".
> - Martin Fowler.

A continuación incluimos distintos aspectos que mejoran la legibilidad del código que consideraremos como *normas de estilo*.

Una *Normas de estilo* o *guía de estilo* es el documento que explica cómo debe escribirse código C en un determinado contexto. Este estilo cambia de una institución a otra, pero en entornos industriales se suele exigir un escrupuloso respeto a estas reglas. A continuación enumeramos las reglas que vamos a exigir en esta asignatura. y que serán *obligatorios* para los ejemplos desarrollados en la asignatura de Programación, en especial para los ejercicios de prácticos y trabajos evaluables.