BLOQUE B: INTR. A LA CIENCIA DE DATOS

3. Bases del discurso: Tipos de datos

La palabra **dato** procede de un antiguo participio pasivo del verbo latino *dare*, o en castellano, dar; por lo que como es lógico, *datum* se podría transcribir como lo dado o lo que se da, por lo que podremos definir dato como **la información que se da sobre un suceso concreto y que permite conocer el conocimiento exacto del suceso o al menos deducir sus consecuencias**.

En el ámbito de la Informática, tiene una acepción bastante más acertada, y es: "**información dispuesta de manera adecuada para su tratamiento por una computadora**", y no sólo eso, sino que también esta palabra ha creado nuevas expresiones dentro de ese ámbito y que hoy en día se escuchan casi cada día en la sección Tecnología de nuestro telediario como "base de datos", "gestión de datos", "inteligencia de datos", "minería de datos", "protección de datos", y un largo etcétera, destacando entre ellos Ciencia de Datos.

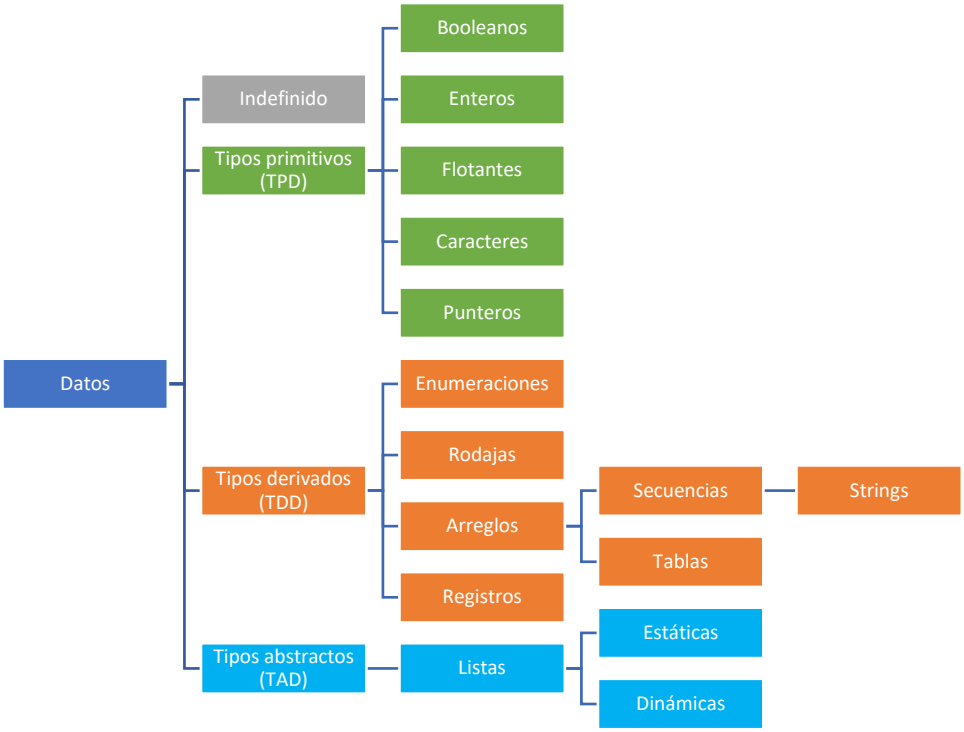
La **Ciencia de Datos** es una de las disciplinas del futuro, más bien es un **campo interdisciplinario** que utiliza diferentes disciplinas como **la estadística, la computación o la algorítmica** para tratar con los distintos datos que se nos pueden presentar en un escenario determinado. Para ello, es muy importante **analizar con detenimiento los datos y estructurarlos**, es decir, clasificarlos.

En la disciplina informática, **el tipo de dato se suele definir como el atributo que indica al programador sobre la clase de datos que se va a manejar**. Esto incluye imponer **restricciones** en los datos, como qué **valores** pueden tomar y qué **operaciones** se pueden realizar.

Conforme a la clasificación de los datos existen tres clases principales:

- ***Tipos primitivos de datos (TPD):*** son datos muy básicos y que forman en sí una **unidad de información atómica**, una unidad mínima de almacenamiento. En esta clasificación pertenecen los **caracteres alfanuméricos**, los **booleanos** (valores de verdad), los **nº enteros**, los **nº reales** y los **punteros**.
- ***Tipos derivados de datos (TDD):*** son datos **creados a partir de la composición de tipos primitivos y estructurados de una forma** determinada. Destacan las **secuencias** o vectores, las **tablas** o matrices, las **enumeraciones**, las **rodajas** y los **registros**. La disposición de los datos primitivos que forman parte del tipo derivado se denomina estructura de dato.
- ***Tipos abstractos de datos (TAD):*** son **datos definidos mediante un modelo matemático y compuestos por un conjunto de datos** más simples **y una serie de operaciones** que se pueden realizar con ellos. **El comportamiento de estos datos no lo define el programador, sino el usuario**. Un buen ejemplo de ello son las **listas**. La implementación de este tipo de datos en un lenguaje de programación se realiza mediante una estructura de datos.

En el contexto de los lenguajes de programación **débilmente tipados**, se podría hablar del **tipo de dato indefinido**. Este tipo de dato surge puesto en este tipo de lenguajes no se declaran previamente las variables, y se declaran inmediatamente tras su primera aparición en el código del programa, pero se tipan una vez se les haya conferido un valor determinado. Si operamos con este tipo de datos, surgirá un error de compilación.



3.1 Tipos Primitivos de Datos

Los **tipos primitivos de datos** se pueden clasificar en:

1. ***Booleanos:*** nombrados así en honor a George Boole, primer matemático que materializó las variables lógicas (que toman sólo el valor de verdadero o falso) a los circuitos eléctricos, **asignando el valor 1 (TRUE) como encendido, y el valor 0 (FALSE) como apagado**. No existen bombillas encendidas y apagadas a la vez ni enunciados atómicos verdaderos y falsos a la vez. Es un **valor discreto y binario que indica la veracidad de una propiedad**. Es un sí o un no, un ying o un yang.
2. ***Enteros:*** se denominan así porque **representan cantidades exactas**. Son los números **utilizados para contar**, por ejemplo para indicar los pisos que tiene un edificio, incluso los negativos, indicándote que se trata de uno subterráneo. Sería un sinsentido tener un rebaño de 6.22 ovejas o un andén 9 y ¾ salvo que vivas en el universo literario de J.K. Rowling. Aunque el concepto de número no existió hasta las primeras civilizaciones, se ha demostrado que varios animales saben contar, lo que indica que **el conteo es propiedad intrínseca de la naturaleza**. Y por si esto fuera poco, en el año 5.000 a.C. en Sumeria se creó la primera computadora del mundo, un ábaco capaz de realizar todo tipo de operaciones, incluso con números negativos, basado eso sí en el conteo de cálculos o piedras pequeñas. Se le considera de hecho el predecesor de la calculadora.

3. ***Flotantes:*** son aquellos números que **no representan cantidades exactas, sino fracciones, partes o porciones** de una unidad como una tarta, una pizza o un **porcentaje** de una muestra poblacional, o una **aproximación de un valor irracional**, el cual no puede representar como fracción. **Los números enteros también están incluidos entre los números reales**, ya que ocho porciones de una pizza formarían la unidad entera, o de manera más formal, cualquier entero se puede representar como el cociente de dos reales, P.ej. 2 = 2/1 = 4/2 = -4/-2 = 1/0.5 = -1/-0.5, etc. Aunque cabe aclarar que **lo que se denomina número real en Matemáticas no corresponde exactamente con lo que se define en Informática como su tipo de dato correspondiente**, puesto la memoria tiene una capacidad finita y no se pueden guardar ni números muy muy grandes ni números muy muy pequeños ni todos los decimales del número π, que de él hasta ahora se han calculado 62 billones, necesitamos una aproximación decimal (lo que implica que se puede representar en formato racional), es lo que se le conoce en Informática como **coma flotante**.
4. ***Caracteres:*** todos aquellos **caracteres alfanuméricos** que no sirven ni para contar, representar porciones, aproximar valores reales o indicar la veracidad de una propiedad son los caracteres propiamente dichos, entre los que incluyen **las letras, los signos, los símbolos y los propios dígitos** del 0 al 9 usados para escribir los números. Conjuntos de caracteres forman la composición íntegra de esta obra.
5. ***Punteros:*** dependiendo del manual del lenguaje de programación con el que se trabaje, a veces se le considera como un tipo de dato primitivo y otras veces no. Realmente es un **"objeto" de referencia cuyo valor se refiere a otro valor almacenado en otra parte de la memoria del ordenador utilizando su dirección**. El puntero **apunta a una ubicación** en memoria, y a **la obtención del valor almacenado en esa ubicación se la conoce como desreferenciación** del puntero. Análogamente, el número de página en el índice de este libro podría considerarse un puntero a la página correspondiente; al ir a la página con el número de página especificada en el índice habremos desreferenciado el puntero. No entraremos en mayores detalles sobre este tipo de dato en concreto, se verá más adelante en el capítulo 10.

Por último, cabe aclarar que los distintos lenguajes de programación pueden considerar más o menos tipos primitivos de datos de los que se exponen según ellos consideren como lo básico.

Por ejemplo, en C no existe como tipo primitivo los booleanos por lo que en su lugar debe usarse un Integer que tome los valores 0 y 1 en sustitución de false y true. De hecho, los lenguajes débilmente tipados no tienen en su gramática el tipo de dato primitivo "puntero". Sin embargo, C tiene otros tipos de dato primitivo como los "unsigned character" (variable que reserva 16 bits de memoria codificados en sistema hexadecimal)

El tipo de dato String o cadena de caracteres se considera primitivo hoy día por la mayoría de lenguajes de programación. Sin embargo, en este libro se considerará como los antiguos lenguajes, como tipo derivado.

Otros lenguajes como Java tienen algún tipo de dato duplicado como los Float, pero cuya diferencia es el espacio de memoria que ocupan, de esta forma "short" indica los reales de capacidad 16 bits, "float" los reales de capacidad 32 bits, y "double" los reales de capacidad 64 bits.

Recordemos que la unidad mínima de información del ordenador es el bit, cuyo valor sólo comprende entre 0 y 1.

A continuación, se exponen las distintas **operaciones que podemos realizar con los distintos TPDs** y cómo los indicaremos en pseudocódigo y en los distintos lenguajes de programación que se desarrollarán a lo largo de este libro.

Con los booleanos, podemos realizar las operaciones lógicas de **negación** (not) (cambiar de true a false y viceversa), **disyunción** (or) (devuelve true si uno de los dos operandos es true) y **conjunción** (and) (devuelve true si y solo si ambos operandos son true) Aquí se muestran las tablas lógicas de las tres operaciones, que indica el valor de la operación según los valores de las variables booleanas a, b:

a	not a
true	false
false	true

a	b	a or b
true	true	true
true	false	true
false	true	true
false	false	false

a	b	a and b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a xor b
true	true	false
true	false	true
false	true	true
false	false	false

Cabe destacar de que ADA tiene una operación extra denominada **disyunción exclusiva** (xor), la cual devuelve true si uno de los dos operandos es true, pero no ambos. Se podría utilizar también en pseudocódigo. Se muestra a la derecha su tabla de verdad.

a xor b equivale a la expresión (a or b) and not (a and b)

Por otra parte, los operadores en C son radicalmente distintos a los anteriormente citados (not, or, and), que son compatibles tanto para pseudocódigo como para Pascal, ADA y Python.

- not a → ! a
- a or b → a || b
- a and b → a && b

El tipo de dato booleano sólo puede tomar dos valores: true y false, que a veces, como en C, ya que en C no existe el tipo booleano, puede escribirse como 1 y 0, respectivamente.

Con los enteros, destacan dos tipos de operaciones. El primer tipo se corresponde con las **operaciones aritméticas**, y estas son **la suma (+), la resta (-), el producto (*), la división entera (/)** (que devuelve el cociente de la división), **el módulo (%)** (que devuelve el resto de la división*), **la exponenciación (^) y el valor absoluto (abs)**. Utilizaremos esta notación para expresar las siguientes operaciones en el pseudocódigo.

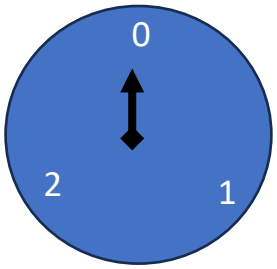
En C, la exponenciación y el valor absoluto no están incluidos en la gramática del lenguaje. A diferencia de C, Pascal utiliza como operadores para la división entera y el módulo (div, mod), respectivamente, y tiene valor absoluto, pero no exponenciación.

Python sí tiene exponenciación, y para ello utiliza el operador **, al igual que en ADA, pero no operador de valor absoluto, cosa que ADA sí tiene. Por otra parte, la división entera la indica con una doble barra (//), que es la que devuelve sólo el cociente de la división. Si utilizáramos la barra simple en Python (/), se convertiría en una división común, lo que hará Python será obtener el resultado exacto de la división, decimales incluidos.

Por otra parte, **ADA realiza una distinción entre el resto de una división y el módulo de un número en una base**, relacionado esto último con la **aritmética modular**, bajo los operadores rem y mod, respectivamente. **Estas operaciones son idénticas si los operandos tienen el mismo signo***, pero si estos tuvieran distinto signo, se distingue mejor la diferencia entre resto y módulo.

Por ejemplo, para x rem 3, los restos de estas divisiones son los siguientes: Recordemos que una división entera a / b debe cumplir la propiedad a = b * q + r, siendo a el dividendo, b el divisor, q el cociente y r el resto, r < b.

x	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
x rem 3	-1	0	-2	-1	0	-2	-1	0	1	2	0	1	2	0	1



Y para x mod 3, se podría asimilar dicha operación como si fuera un reloj de tres horas; 0, 1 y 2 y mover la aguja empezando desde el cero x veces hacia la derecha si el número es positivo y x veces hacia la izquierda si el número es negativo. La hora que marque la aguja será el resultado de la operación.

x	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
x mod 3	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1

Luego, **las operaciones "a rem b" y "a mod b" dan el mismo resultado cuando "a, b" tienen el mismo signo o "a" sea múltiplo de "b"**. Por ello, cuando nos pidan por ejemplo un número par, la condición que debe cumplir puede ser perfectamente o "a rem 2 = 0" o "a mod 2 = 0", o en otros lenguajes, "a%2 = 0"

Cabe destacar que **no todas las operaciones aritméticas que se nos ocurran** (p.ej. raíz cuadrada, raíz quinta, logaritmo, seno e incluso exponenciación en algunos lenguajes etc.) **están disponibles en los lenguajes de programación**, por lo que para utilizar una operación no existente en dichos lenguajes **debemos o bien utilizar un algoritmo** que defina dicha operación, **o bien instalar el paquete de recursos en el cual esa operación está incluida como una función** (p.ej. sqrt(), exp(), log(), sin(), etc.), suelen tener el nombre de Maths o Numerics. Se especificará siempre en el enunciado si hay que importar alguna librería y cómo se hace según el lenguaje de programación que se use.

El segundo tipo de operaciones corresponde con las **operaciones relacionales**. En pseudocódigo y en Pascal, estas son: **mayor que** (>), **mayor o igual que** (>=), **menor que** (<), **menor o igual que** (<=), **igual a** (=) y **distinto de** (<>)

En el resto de lenguajes, los cuatro primeros operadores son idénticos, para la operación "distinto de", su operador en ADA es (/=), mientras que en Python y C es (!=); y para la operación "igual a", su operador en Python y en C es (==)

Esto es debido a que la asignación en estos dos lenguajes utiliza como operador (=), a pesar de encontrarse con la oposición de varios programadores, ya que cuando se asigna un valor a una variable, no se iguala la variable a dicho valor, sino que guarda en una dirección de memoria con el nombre de esa variable ese valor.

Luego, hay que tener cuidado de no equivocarse con la operación "igual a", la cual devuelve siempre un valor booleano, y la asignación. Si la comparación de igualdad se expresa mediante el operador (=), entonces el operador de asignación será (:=), es lo que sucede con Pascal y ADA. Si pasa al contrario como en Python y C, en el que el operador (=) es el de la asignación, entonces, la operación "igual a" tendrá como símbolo (==)

Matemáticamente, el conjunto de los números enteros es infinito, abarca desde el – ∞ hasta el + ∞, pero en la Informática, **el tipo de dato entero está acotado entre dos números**: uno muy muy pequeño y otro muy muy grande**. Esto es debido a que la memoria tiene una capacidad finita** y no se pueden almacenar ni números muy muy muy pequeños ni números muy muy muy grandes.

Estos extremos están por defecto etiquetados en los lenguajes de programación. Por ejemplo, para Pascal, [MinInt, MaxInt]; para Ada, [Integer'First, Integer'Last]; para Python, [sys.minint, sys.maxint] (debe haberse instalado previamente la librería sys): y para C, [INT_MIN, INT_MAX] (debe haberse instalado previamente la librería limits.h)

Dependiendo de la **arquitectura del procesador** (es decir, el número máximo de bits que puedo almacenar en memoria para el valor de una variable), el valor de estos extremos es para MinInt = -2^31 o -2^63 y para MaxInt = 2^31 – 1 o 2^63 – 1, según la arquitectura de 32 bits y la de 64 bits, respectivamente.

Como dato adicional, no viene mal saber que **se pueden utilizar barras bajas (_) como separador de dígitos**, realizando la misma función que el punto que separa millares en español, o la coma en inglés. En lugar de escribir 1,234,567 podemos escribir 1234567 o 1_234_567.

Con los flotantes, o vulgarmente dicho, los reales, **podemos usar todas las operaciones que se habían definido anteriormente para los enteros, a excepción de la división entera y el módulo, que se sustituyen por la división normal (/)**. Esta división da el resultado exacto con precisión decimal de dividir a entre b, devolverá un número real k que multiplicado por b diera como resultado a.

Conforme a su representación, surge el mismo problema de memoria, pero esta vez **deben guardarse tanto los dígitos de parte entera como los dígitos de la parte decimal** de nuestro número real, y distinguirse entre sí. **Esta manera de representar reales por los valores de sus parte entera y de su parte decimal se denomina coma flotante**, puesto en realidad, el número se representa a través de una mantisa (el número decimal sin la coma), multiplicado por una potencia de base 10 elevada a un exponente determinado. Este exponente decide cuántas posiciones a la izquierda o al a derecha se desplaza la coma con respecto al último dígito donde termina la parte entera del número, de forma que quede sólo un dígito del 1 al 9 delante de la coma (cuidado, en inglés, nuestra coma

decimal se representa mediante un punto), de **una forma parecida a la notación científica** en la calculadora.

P.ej. 3234.5 → 3.2345E+3, 0.00016 → 1.6E-4, que sustituyen a las formas de representación matemáticas 3.2345 * 10^3 y 1.6 * 10^-4, respectivamente.

Evidentemente, podemos escribir cualquier número decimal usándolo como es en su esencia, con su parte entera, su coma decimal y su parte decimal. Sin embargo, hay que tener en cuenta de que **si por alguna razón necesitáramos asignarle un valor entero a una variable flotante, es obligatorio escribir el número con la coma decimal** para indicar que su tipo de dato es el flotante y no confundirlo con un entero. Luego, en lugar de asignar a una variable el valor 5, se le asignará el valor 5.0.

Como máximo, a la hora de asignar valor a las variables flotantes, se pueden utilizar números de hasta 24 dígitos (de los cuales 7 son de la parte decimal) para sistemas de 32 bits; y números de hasta 53 dígitos (de los cuales 15 son parte decimal) para sistemas de 64 bits.

Finalmente, tenemos **los caracteres**, que abarcan desde espacios en blanco, puntos, comas, signos, símbolos, hasta las letras que comúnmente conocemos del abecedario latino, de la A a la Z tanto en minúsculas como en mayúsculas.

Las operaciones que se pueden realizar con caracteres **son más limitadas, y no son nada similares a las operaciones matemáticas que conocemos**, por lo que es cuestión de práctica el tener que acostumbrarse a trabajar con caracteres.

Para ello, se liga mediante un **código internacional** llamado **ASCII** cada carácter con un valor numérico determinado. Un anexo aparte mostrará algunos de los **256 caracteres ASCII los cuales están ligados a un índice cada uno**. Así por ejemplo, la letra 'A' está ligada al índice 65.Es importante fijarse en esta relación entre carácter e índice (no memorizarla, se dará el correspondiente ASCII del carácter en el enunciado), ya que entonces, se entenderá mucho mejor la aritmética de caracteres. **A la hora de denotar caracteres, los denotaremos entre comillas simple**s, para no confundir la variable a con el carácter 'a'

En primer lugar, viene bien saber que **se pueden utilizar operaciones relacionales en todos los lenguajes de programación. El orden de caracteres lo establece el propio código ASCII**, y tiene la ventaja de ser intuitivo, puesto **se sigue el orden numérico para los dígitos y el alfabético para las letras**, tanto mayúsculas como minúsculas, aunque siendo "mayores" las letras minúsculas al aparecer después de las mayúsculas.

Algunos enunciados booleanos verdaderos con estos operadores relacionales son p.ej. 'a' = 'a'; 'a' <> 'A'; 'a' > 'A'; 'A' < 'a'; 'b' < 'g'; 'z' > 'm'; '3' < '4'; '5' > '0' y otros enunciados menos intuitivos como '!' < '+'; '&' > '$'; ''' <> '"'; '2' > '%'; '7' < '^'; '0' < 'o'; 'D' < 'c': 'd' > 'c'; ' ' <> ''.

Otras operaciones que pueden aplicarse a los caracteres son: **el ordinario de un carácter** (ord()), que devuelve el índice ASCII ligado al carácter; su inverso, **el carácter de un número** (chr()), en el que dado un número dentro del rango de índices ASCII, devuelve el carácter correspondiente según el índice; las operaciones booleanas **isAlpha(), isDigit(),** en las que dado un carácter, indica si este es una letra del alfabeto o un dígito numérico, respectivamente; y las operaciones **upper(), lower(),** que

transforman una letra del alfabeto a mayúscula y a minúscula, respectivamente. En algunos lenguajes y en pseudocódigo, existen dos operaciones extra válidas**, succ() y pred(),** en las que dado un carácter, devuelve el siguiente y el anterior carácter al dado, respectivamente.

Por ejemplo, ord('B') = 66, chr(66) = 'B'; isAlpha('B') = true; isDigit(`B`) = false; upper('B') = 'B'; lower('B') = 'b'; succ('B') = 'C'; pred('B') = 'A'.

Evidentemente, existen bastantes más caracteres que los dados, pero nos vamos a abstener a utilizar exclusivamente los caracteres del inglés, ya que **existen muchos lenguajes que no aceptan caracteres especiales**, es decir, caracteres de otras lenguas distinta a la inglesa. Luego, a la hora de crear nuestros programas, **debemos procurar no utilizar caracteres especiales, lo que incluye caracteres comunes del español como letras con tilde, la u con diéresis y la letra eñe**, por lo que les suprimiremos sus tildes, diéresis y virgulillas. Así, en vez de escribir á, é, í, ó, ú, ü, ñ; escribiremos a, e, i, o, u, u, nn**. Los signos de apertura de interrogación (¿) y de exclamación (¡) simplemente se omiten**.

Conforme a los Strings, en este libro se les considerarán como una secuencia de caracteres y no como un TPD, por lo que se impartirán en el Tema 8: Arrays.

Los punteros también son otro TPD, pero se impartirán mucho más adelante, cuando se usen en nuestros programas más sofisticados y con estructuras de datos más complejas.

Las distintas operaciones para los distintos TPDs **pueden combinarse entre sí siempre y cuando los dos operandos resultantes no sean de distinto tipo**. P.ej, 'A' + 3 es una operación inválida pero ord('A') + 3 sí lo sería, pues el ordinario de 'A' devuelve un resultado del mismo tipo de dato que 3, un entero. De la misma forma, 3 + 5.2 sería inválido, pero 3.0 + 5.2 sí lo sería. **También sería inválido operar con un tipo de dato que ese operador concreto no admite**, por ejemplo, expresiones como succ(true), 'd' – 'c', 3 or 4 son inválidas.

La **jerarquía de operaciones** que se sigue al operar expresiones combinadas es la siguiente:

1. Paréntesis
2. Exponenciación y funciones matemáticas
3. Operadores de caracteres (y enumeraciones, rodajas)
4. Desreferenciación (punteros y accesos de arreglos o registros)
5. Negación
6. Producto, división, módulo y conjunción
7. Suma, resta y disyunción
8. Operadores relacionales
9. De izquierda a derecha

P.ej:	(5 * (3 + 4) = ord('B') / 2 + 2) or not (5^2 > sqrt**(ord('d'))** and isDigit**(chr(50)**))
Paréntesis	(5 * 7 = ord('B') / 2 + 2) or not (**5^2 > sqrt(100)** and isDigit('2'))
Exponenciación y funciones mat.	(5 * 7 = **ord('B')** / 2 + 2) or not (25 > 10 and **isDigit('2')**)
Operadores de caracteres	(5 * 7 = 66 / 2 + 2) or not (25 > 10 and true) = (5 * 7 = 66 / 2 + 2) or **not true** {Por preferencia del paréntesis}
Negación	(5 * 7 = 66 / 2 + 2) or false
Producto, división, módulo y conjunción	(35 = 33 + 2) or false {La disyunción depende del valor de la expresión 35 = 35}
Suma, resta y disyunción	35 = 35
Operaciones relacionales	true {Resultado final}

Para finalizar, se muestran a modo de resumen los distintos operadores que hemos visto para manejar diferentes TPDs para pseudocódigo y para los lenguajes de programación con los que trabajaremos en otro anexo aparte.

3.2 Tipos Derivados de Datos

Los **tipos derivados de datos** se pueden clasificar en:

1. ***Enumeraciones:*** es un tipo de dato compuesto por varios elementos llamados **enumeradores o identificadores**, los cuales están **etiquetados**. Por ejemplo, una enumeración de los posibles mazos que pueden obtenerse en una carta de la baraja española es Mazos = (Oros, Copas, Bastos, Espadas), y la de sus posibles valores es Valores = (As, 2, 3, 4, 5, 6, 7, Sota, Caballo, Rey) Como se puede observar, **la enumeración actúa como una variable qué solo puede tomar como valores un conjunto de elementos** ordenado o no, finito y determinado por el usuario, por ello, a los TDDs se les conoce también como tipos definidos por el usuario. **Esto confiere mayor seguridad a las variables**, ya que si por ejemplo, declaro Sexo como el TPD carácter, se espera que el usuario introduzca 'M' para masculino y 'F' para femenino, pero puede introducir otro carácter distinto que no nos interese. Por ello, declaramos Sexo como el TDD enumeración, para limitar el nº discreto de valores que puede tener una variable de ese tipo a M, F (ojo, aquí son etiquetas, no caracteres). Si entonces se introdujera otra etiqueta, entonces se imprimirá en pantalla un mensaje de error avisándole al usuario de que se ha equivocado. Normalmente, las enumeraciones agrupan una serie de identificadores que comparten algo en común, p.ej. Sexo = (Masculino, Femenino), Estaciones = (Otoño, Invierno, Primavera, Verano) Además, podemos utilizar como etiquetas un número, una letra, una palabra o una serie de palabras, a nuestra libre disposición, llegando a mezclar dichos formatos de etiquetado incluso para la misma enumeración.
2. ***Rodajas:*** aunque sólo están disponibles como TDD ya implementado para algunos lenguajes, en este libro introduciremos este concepto a nuestro pseudocódigo ya que es una óptima herramienta para los principiantes en la programación que ayuda a simplificar el código. En

otros manuales, esta herramienta no se reconoce y se opta por diseños más formales y concurrentes típicos de la mayoría de lenguajes. La rodaja es un tipo de dato que **deriva de un rango de valores perteneciente a un tipo primitivo**. El nombre rodaja sugiere como si de tener un solomillo y tomar la parte del solomillo que más nos gustara se tratara. **En este símil, el solomillo entero es el TPP**, mientras que **la porción de solomillo** que queremos servir en nuestro plato **es la rodaja**. Por ello es que también se le conoce como **subrango o subtipo**. Por ejemplo, el conjunto de los nº naturales es una rodaja o subconjunto del conjunto de los nº enteros. Sabemos que todos los naturales son enteros, pero no todos los enteros son naturales. Por lo tanto, decimos que los naturales son un TDD rodaja del TPD entero. Así mismo, el número 3 pertenece tanto al TPD entero como al TDD natural, y por lo tanto, **las operaciones que se puedan realizan en el TPD también pueden realizarse sobre el TDD siempre que su resultado quede dentro del rango**. Las rodajas se indican bajo la notación a..b, siendo a el primer valor del subrango y b, el último. De esta forma, siendo el rango de los enteros int = MinInt..MaxInt, el subrango de los naturales se denotará como nat = 0..MaxInt. IMPORTANTE: aplica para rangos discretos, por lo que **no se admiten rodajas derivadas del tipo de dato flotante**.

3. ***Arreglos:*** es un tipo derivado que almacena **una serie de valores contiguos del mismo tipo**. Se subdividen en dos tipos principales: las **secuencias o vectores**, cuando este arreglo está organizado en una única fila como por ejemplo el vector u; y las **tablas o matrices**, cuando este arreglo está organizado en filas y en columnas como por ejemplo la matriz A. Sus nombres de variable se denotan con uno o dos parámetros de dimensión encerrados entre corchetes a su derecha: u[n] y A[m, n], siendo m el número de filas que posee el arreglo (en el caso del vector, m=1, se omite) y n el número de columnas. A la hora de asignar los valores del arreglo, estos deben ir entre corchetes y separados por comas. En el caso de las matrices, cada fila debe ir separada por punto y coma. En este ejemplo, u[3] = [3, 1, -1] y A[3,2] = [1, 2; 0, -1; 3, 1] Debajo se muestran lo que serían sus equivalentes en notación matemática:

$$u = (3 \quad 1 \quad -1); A = \begin{pmatrix} 1 & 2 \\ 0 & -1 \\ 3 & 1 \end{pmatrix}$$

4. ***Registros:*** es un tipo de dato derivado que almacena **una serie de valores contiguos pero que estos pueden ser de diferente tipo**. Este conjunto de valores denominados **campos forman una estructura de datos** que definen un objeto cotidiano. Son **uno de los tipos más comunes que nos podemos encontrar como representación de conceptos más complejos**. P.ej. un documento de identidad, la etiqueta de una prenda, el ticket de la compra, una carta de la baraja española, etc. Para declarar un nuevo tipo de registro, se denota escribiendo cada campo entre llaves, separados por comas. P.ej. ID = {Nombre, Apellidos, Num, Letra}, y para crear un objeto de ese tipo, lo mismo pero dando ya los valores de esos campos. P.ej. ID1 = {“Carmen”, “Muestra Muestra”, 123456789, ‘A’} Como se puede observar, las comillas son imprescindibles, puesto diferencian claramente entre lo que es el nombre del campo, y el valor y tipo de ese campo.

A continuación, se exponen las **operaciones de conjunto** que se pueden realizar con los TDDs enumeración y rodajas, así como con cualquier TPD. Trataremos más adelante y más detalladamente con los TDDs arreglos y registro en sus temas, el 8 y el 10, respectivamente.

Dados los TDDs enumeración Días = (L, M, X, J, V, S, D) y rodaja Alfabeto = 'A'..'Z' (compuesta por los caracteres ASCII que pertenecen al alfabeto y son mayúsculos), las operaciones que se pueden realizar son las siguientes:

- ***Operadores relacionales:*** de la misma forma que 'A' < 'F' (al aparecer A antes que F en el alfabeto y en la lista de caracteres ASCII), y 'W' > 'E', con las enumeraciones podemos hacer lo mismo, por lo que L < X, V > M, S <> D, son enunciados correctos. Aunque cabe añadir que no en todas las enumeraciones tiene sentido hablar de orden, p.ej. en la enum. Mazos anterior tiene lógica para el computador enunciados como Oros < Espadas o Espadas > Bastos, porque el orden con el que hemos declarado esta enumeración es Mazos = (Oros, Copas, Bastos, Espadas), pero el orden de mazos en realidad no tiene ningún fin práctico. En el caso de Días, tiene su fin porque sigue el orden de los días de la semana, empezando por el Lunes.
- ***Extremos:*** se realiza bajo las operaciones **first()** y **last()**, que devuelven respectivamente el primer y último valor del conjunto de valores que forma el tipo de dato. Por ejemplo, first(int) = MinInt, first(nat)= 0, first(char)='', first(Días) = L, first(Alfabeto) = 'A', last(Días) = D, last(Alfabeto) = 'Z', last(Mazos) = Espadas.
- ***Posicionamiento:*** la operación **pos()** devuelve la posición relativa de un valor de la enumeración o de la rodaja en el conjunto que lo define. Por ejemplo, pos(L) = 1, pos(S) = 6, pos(V) = 5, pos(Bastos) = 3, pos('L') = 12. Por otra parte, su operación inversa es **val()**. Así para el tipo de dato Día, val(1) = L, val(6) = S, val(3) = X; para el tipo de dato Alfabeto, val(12) = 'L', val(5) = 'E'.
- ***Anterior y posterior:*** se utilizan las operaciones **pred()** y **succ()**, que devuelven respectivamente el anterior y el posterior de un valor. P.ej. pred(0) = -1, succ(0) = 1, pred('C') = 'B', succ('P') = 'Q', pred(S) = V, succ(X) = J, pred(Bastos) = Copas, succ(Oros) = Copas. Hay que tener **especial cuidado con no introducir como operando un extremo del tipo**. Para cualquier tipo T, las expresiones pred(first(T)) y succ(last(T)) darán lugar a un error de desbordamiento. De este modo, expresiones como pred(L) o succ(D) son ilegales.
- ***Pertenencia:*** El operando **(in)** recibe como operandos un valor y el nombre de un tipo de dato y **devuelve un booleano que indica si ese dato pertenece a dicho tipo de dato**. P.ej. estos enunciados que utilizan operador de pertenencia son verdaderos: 3 in nat, 3 in int, -1 not in nat, -1 in int, '&' in char, 'G' in char, '&' not in Alfabeto, 'G' in Alfabeto, Bastos in Mazos, Corazones not in Mazos, J in Días, W not in Días.

Las principales diferencias entre estos cuatro TDDs radican en si se permiten que los tipos de dato que componen estos sean distintos o no y en si almacenan sólo un valor de su rango (valores discretos) o más de un valor (valores contiguos).

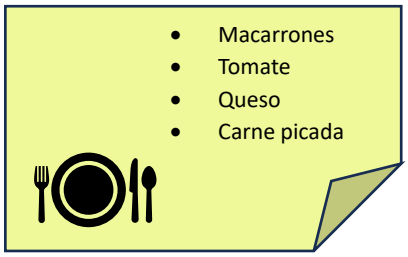
	Valores discretos	Valores contiguos
Valores del mismo tipo	Rodajas	Arreglos
Valores de distinto tipo	Enumeraciones	Registros

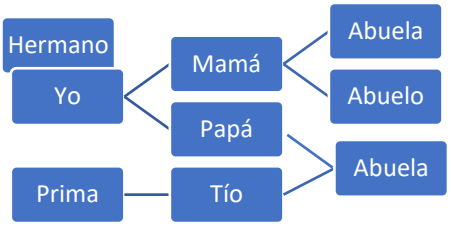
3.3 Tipos Abstractos de Datos

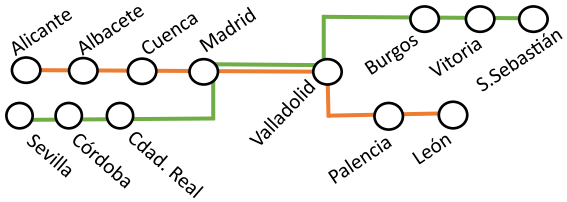
El resto de **estructuras de datos definidas por modelos matemáticos**, siendo estos **definidos a la vez por el punto de vista del usuario** y cómo operan con dichas estructuras, son los denominados **TADs**.

No vamos a desarrollar este tipo de datos en este curso ya que entonces complicaríamos bastante el temario, y este curso se supone que debe introducirnos al mundo de la programación.

Simplemente se deben dar a conocer este tipo de datos para hacer la siguiente objeción: "**Toda forma de organización de los datos que al ser humano se le ocurra es inherentemente un tipo abstracto de dato**" Como ejemplos, se muestran aquí los más comunes, una lista de la compra, un árbol genealógico, un plano de líneas de transporte, la fila del supermercado, etc.







Nos centraremos únicamente y de forma superficial en las **listas**, al ser la estructura más parecida a la secuencia, en el tema 11.

3.4 Ejercicios propuestos

Se sugieren realizar los siguientes ejercicios para comprobar que se han comprendido los conceptos más importantes de este tema. Aunque estos ejercicios no sirvan para programar como tal, sí que forman parte de unos conocimientos previos que hay que tener dominados antes de empezar a programar de verdad.

1. Dadas las siguientes variables a = true, b = false, c = 6, d = -2, e = 0, f = 5, g = 2, h = 0.0, i = 1.0, j = 'y', k = 'e', l = '2', m = '@'; evalúe las siguientes expresiones. Si la expresión produce un error, indícalo y por qué:
   - (*) not a
   - (*) a or b
   - (*) a and b
   - (*) a and not b
   - (*) a + b
   - (**) upper(j)
   - (**) pred(k)
   - (**) isDigit(k)
   - (**) succ(m)
   - (**) c * d + f / g

- (*) f + g
- (*) c – d
- (*) d * f
- (*) f / g
- (*) f / e
- (*) c % f
- (*) i % -d
- (*) g + i
- (*) 3.0 * i
- (*) h + 2
- (**) ord(j)
- (**) ord('j')
- (**) isAlpha(e)
- (**) isAlpha('e')
- (**) isAlpha(l)
- (**) upper(l)
- (**) 2.5 * (2 .0 + i)
- (**) a > b
- (**) e < f
- (**) g + c >= 5
- (**) (i + h)/2.0 > 0.5
- (**) 'y' <> j
- (**) m > k
- (**) k < j
- (**) 'k' < 'j'
- (***) (2*c % f = g) xor isDigit(l)
- (***) succ (chr(ord('j') + g))
- (***) upper(pred(k))
- (***) isAlpha(j) and (k < succ('k'))
- (***) 3.5*i + 2.0 > c - d
- (***) not isDigit(chr(ord('+') + f))

- (****) c + d / g – (ord(k) – ord (m)) / c + ord(succ(chr(sqrt(10*g + f) + f*c)))
- (****) (-f^g > -ord(m)) and isAlpha(chr(10^-d + ord(l) + c*d – ord(pred('2'))))

64. Dadas las siguientes variables del tipo enumeración Días = (L, M, X, J, V, S, D): Hoy = X, Ayer = L y de los tipos rodaja Positivo = 1..MaxInt : n = 1, y Alfabeto = 'A'..'Z' : Letra = 'E'; evalúe las siguientes expresiones. Si la expresión produce un error, indícalo y por qué:
    - (*) pred(Hoy)
    - (*) pred(Ayer)
    - (*) succ(Hoy)
    - (*) succ(Ayer)
    - (*) isAlpha(Letra)
    - (*) lower(Letra)
    - (*) n – 1
    - (**) Letra > 'D'
    - (**) Letra < J
    - (**) L < J
    - (**) 'L' < 'J'
    - (**) n in int
    - (**) 'T' in Alfabeto
    - (**) 't' in Alfabeto
    - (**) 'X' in Días
    - (**) X in Días
    - (***) pos('S')
    - (***) pos(S)
    - (***) n + ord(Letra)
    - (***) val(ord(Letra) / 12) para Días
    - (***) val(ord(Letra) / 12) para Alfabeto
    - (***) first(Alfabeto) = Letra
    - (***) last(Alfabeto) <> 'z'
    - (****) pred(val(pos(Hoy) - 1)) para Alfabeto

4. Instruir el discurso: Asignaciones

Empezamos a continuación con lo que se podría denominar la parte nuclear de la asignatura, que comprende básicamente los capítulos 4-8, en donde **diseñaremos ya nuestros primeros programas** totalmente funcionales en los distintos lenguajes de programación. Para ello, usaremos a lo largo de este curso varias herramientas de software para los varios lenguajes de programación ya mencionados.

No obstante, cabe destacar que este libro no utiliza la metodología clásica en la que se enseña el lenguaje de programación como un idioma del que se debe obtener un título de nivel desde A1 a C2. Personalmente, esto lo considero un error muy grave, ya que merma la capacidad de razonar del futuro programador, pues se tiende a mecanizar los problemas, ya que en la metodología clásica, se van enseñando poco a poco las sentencias que se pueden dictar con el lenguaje de programación, para qué sirven y cómo aplicarlas, desde conceptos básicos casi comunes hasta conceptos muy nativos pero qué sólo funcionan en ese lenguaje.

El objetivo de esta obra es poder acercarnos a hablar con del ordenador, no tanto su idioma, sino acercándonos a **cómo piensa el ordenador**; por ello el subtítulo no dice "guía para hablar", sino "guía para discursar". Los lenguajes de programación (idiomas) son en realidad implementaciones de los algoritmos (ideas) que hemos estado diseñando a lo largo de esta obra, y el **pseudocódigo** y el **diagrama de flujo** son como dos lenguajes universales comprensibles a nivel humano alrededor de todo el globo terráqueo y que **simulan el pensamiento computacional**, pero **el ordenador no es capaz de comprenderlo**.

La **Programación** es uno de los **tópicos más importantes de la Ingeniería Informática**, así como la **base mínima** que en este siglo de la **digitalización** todo ingeniero debe tener, por no decir que es la llave que abre las puertas entre el mundo exterior y el ordenador. Todo uso que se te ocurra de las Nuevas Tecnologías, en especial de las TICs, o las **Tecnologías de la Información y de la Comunicación**, necesita programas.

Para poder resolver estos grandes problemas, no se consta ni de mecanizar ni de copiar y pegar métodos de los grandes programadores, **la Programación consiste en resolver a través del razonamiento de un ordenador y de forma razonada un problema**, siguiendo las normas gramaticales de un lenguaje determinado y explicar por qué se ha resuelto en esa serie de pasos determinado. De nada sirve talar un bosque si no se sabe hacer leña. Y es que al fin de al cabo, todos los caminos llevan a Roma, puede existir más de una resolución al problema. Cualquier problema que se nos ocurra, lo puede resolver nuestro ordenador siempre y cuando sepamos qué hay que hacer y se lo indiquemos en una forma que lo comprenda, como a un gólem cuando le introducimos un papel.

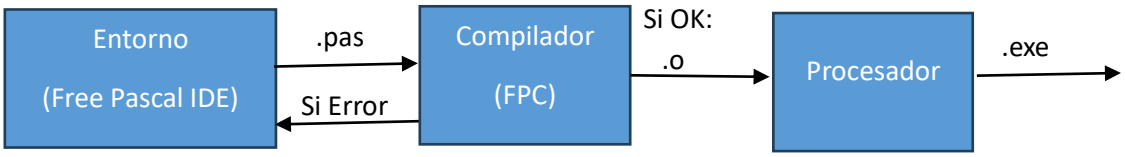
Asentemos a continuación nuestras bases. ¿Cómo podemos hacer un programa totalmente funcional y que lo comprenda el ordenador?

En primer lugar, debemos conocer en qué ambientes puede un ordenador ser capaz de reconocer un programa escrito en un lenguaje y procesarlo. Hablamos de los **compiladores** y los **entornos de programación**.

Los entornos son **aplicaciones idóneas donde podremos crear archivos con una extensión la cual el compilador puede leer su contenido** (que es nuestro programa codificado en ese determinado lenguaje de programación) **y traducir las instrucciones de dicho programa al lenguaje máquina** para que el procesador de nuestro ordenador ejecute la tarea encomendada.

Cabe diferenciar entre entorno de programación, que es donde se **crea el archivo de nuestro programa** y se envía al **compilador**; y este último, que es lo que **lee nuestro archivo** y **transforma** el conjunto de instrucciones de nuestro lenguaje de programación **al lenguaje máquina** y lo envía al procesador. Muchas veces, los compiladores no se incluyen en la instalación del entorno, y deben instalarse aparte. Por ejemplo, para ADA debemos instalarnos el compilador GNAT y el entorno AdaGide, y para Python y C debemos descargarlos dentro de la propia aplicación de VS Code.

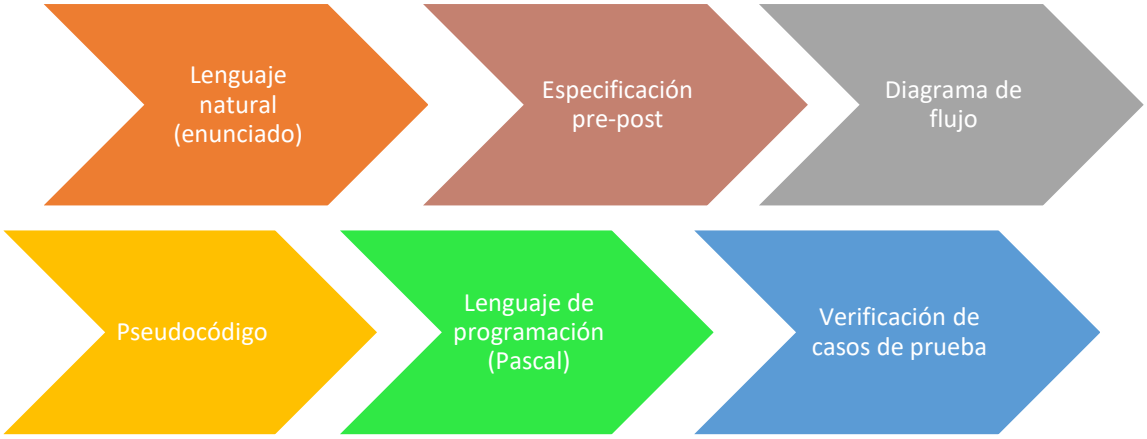
Nosotros usaremos a partir de ahora por defecto el lenguaje Pascal. Para programar en este lenguaje, basta con descargar Free Pascal, que incluye ambos, entorno de programación (Free Pascal IDE) y compilador (FPC).



En este entorno, codificaremos nuestros programas en el lenguaje Pascal, y los guardaremos en la extensión propia del lenguaje, pas. Además, este entorno permite ejecutar nuestros programas. Para ello, el entorno envía nuestro archivo pas al compilador. Si todo es correcto, el compilador crea nuestro código máquina de extensión o y lo envía al procesador, que ejecutará el programa, creando un ejecutable (.exe). En caso contrario, el compilador enviará un mensaje de error y la línea del documento en la que se ha detectado.

A modo de anexo, se adjuntarán guías breves para programar en los cuatro lenguajes que usaremos en este curso: Pascal (por defecto), ADA, C y Python.

Vamos a diseñar nuestro primer sencillo programa, será un programa en el que dado el radio de un círculo, calcule el área de este. Su fórmula es: $A = \pi r^2$, siendo el valor de π (pi) aproximadamente 3.1416. Como principiantes en la programación, se sugiere seguir la siguiente serie de **pasos para** poder **diseñar cualquier** tipo de **programa**:



1. **Leer y comprender el enunciado**, qué se pide, qué datos de entrada tenemos y qué datos de salida se esperan. Detectar también posibles ambigüedades.
2. Realizar la **especificación pre-post** del programa, los tipos de datos esperados, qué condiciones se esperan que cumplan los datos de entrada y los datos de salida.
3. Diseñar un buen **diagrama de flujo** ayuda a visualizar mejor las **ideas principales** del algoritmo. Establece la serie de pasos que hay que realizar para ejecutar el problema en un borrador u hoja en sucio y después expresa esos pasos en bloques y sus transiciones en líneas de flujo.

4. Observando las **estructuras de control** que contiene el diagrama de flujo, podemos fácilmente intuir el **algoritmo en pseudocódigo**. En el pseudocódigo, podemos utilizar todas las herramientas que se nos ocurran, independientemente de si estas son soportadas por el lenguaje solicitado o no.
5. Finalmente, **desarrollamos el programa** cumplimentando con la estructura similar al texto instructivo que hemos visto: cabecera del título, especificación, declaraciones e **implementación del algoritmo en ese lenguaje de programación**. En este paso, nuestras herramientas son limitadas y **debemos ajustarnos a las reglas gramaticales** del lenguaje.
6. No viene mal agrupar en una tabla una serie de **casos de prueba significativos** y sus resultados esperados y **ejecutar el** nuevo **programa** confiriendo esos valores a los datos de entrada para comprobar que **se obtienen los resultados esperados**.

Esto es sólo una **metodología básica e inicial** que se sugiere usar con el fin de ayudar al joven programador a diseñar sus programas correctamente, especialmente aquellos más complejos. En cuanto avancemos más, podremos irnos más directos al grano, al paso 5, pero al menos en esta parte nuclear de la asignatura es altamente recomendable seguir estos pasos, sobre todo si eres nuevo en el mundo de la programación.

Recordamos que un programa es correcto cuando **coincide lo que se obtiene de valor salida al ejecutar el programa y lo que se debería obtener según la especificación** del programa, por ello es muy importante hacer hincapié en realizar buenas especificaciones y en procurar seguir el diseño del programa según ese contrato. Un programa que no sigue su especificación no es correcto.

Además, recuerda que todos los caminos llevan a Roma, por lo que **dos programas distintos pueden resolver el mismo problema correctamente**, y por tanto los dos son iguales de válidos. Su diferencia radica simplemente en la sencillez con la que se puede seguir el diseño del programa (longitud de las instrucciones, complejidad de las estructuras de control, nº de variables a declarar, etc.) o eficiencia. Entre todas las características de los programas, **Bases de la Programación priorizará la corrección**.

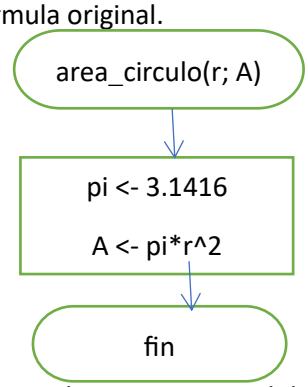
A continuación, procedemos a diseñar el programa propuesto del área del círculo.

El área de círculo, que es lo que queremos obtener, recibe como única incógnita el radio del círculo, único dato de entrada. El enunciado no es ambiguo, pues aunque no se ofrece la unidad de medida del radio, la unidad que el usuario decidiera tomar no afectaría en absoluto al resultado del área.

El tipo de dato radio puede ser o bien entero o bien real, pero asumiremos que r es real porque es el tipo de dato menos restringido de ambos. La especificación pre-post del programa es la siguiente:

- Datos de entrada: un real r, radio de la circunferencia
- Precondición: r > 0.0
- Datos de salida: el área del círculo, A, nº real
- Postcondición: A = 3.1416 * r^2

Cabe recalcar que **si no se menciona nada sobre entrada por teclado o salida en pantalla, los datos de entrada y salida se obtienen desde memoria**, y será así hasta dar el tema 7. Procedemos a continuación a diseñar su respectivo diagrama de flujo y posteriormente, el pseudocódigo. En este caso se trata de un algoritmo muy sencillo, ya que tan sólo hay que aplicar una fórmula matemática. No obstante, para optimizar mejor el código, vamos a declarar una variable PI <- 3.1416, siendo más fieles a la fórmula original.




```
area_circulo(r; A){
  pi := 3.1416;
  A := pi * r^2;
}
{ En pseudocódigo, deben indicarse los datos
de E/S estándar o de memoria que se toman }
```

En este caso observamos que el diagrama de flujo sigue una **estructura secuencial**, lo que quiere decir que **ejecuta las instrucciones una a una, en orden y obligatoriamente**. Vamos a ver a continuación cómo desarrollar un programa en estructura secuencial.

En primer lugar, dos cosas comunes a todos los programas son el **título del programa y la especificación** de este. El título debe tener un nombre que al leerlo nos dé una idea de para qué sirve el programa. La especificación preferiblemente debe ser la pre-post, y debe introducirse a modo de comentario. Ambas partes del programa dependen por supuesto del lenguaje de programación que utilicemos.

En Pascal, la cabecera del programa debe ir precedida de la palabra reservada program, y sucedida por un punto y coma (;) y los comentarios se introducen entre dos llaves. El inicio de nuestro programa en Pascal debería ser el siguiente:

```
program area_circulo;

{ Datos de entrada: un real r, radio de la circunferencia
Precondición: r >0.0
Datos de salida: el área del círculo, A, nº real
Postcondición: r = 3.1416 * r^2 }
```

Posteriormente, debemos **declarar toda variable o tipo no primitivo** que utilicemos, lo que nos abre una nueva sección en este tema. Estas declaraciones actúan como si fueran una lista del **inventario** que hay que tener a mano para seguir una receta o una tarea.

4.1 Declaraciones

En los lenguajes de programación **de alto nivel**, cuando se quiera utilizar una variable, **es obligatorio indicar anteriormente el tipo de dato que se va a usar con ella**, como si fuera un listado de las cosas que hay que tener antes de ejecutar una tarea. Como se ha observado **en el pseudocódigo**, a diferencia de en el programa, **no es obligatorio declarar variables**.

La manera en la que se realizan las declaraciones vuelve a depender por supuesto del lenguaje de programación. Además, cabe distinguir entre constantes y variables.

Las constantes se declaran al principio del programa y, como su nombre indica, tienen la peculiaridad de que, **una vez declarado su valor, lo mantienen durante toda la ejecución** del algoritmo. Si el algoritmo intentara por lo que fuera manipular el valor de una constante (p.ej. se intenta ejecutar la instrucción pi:=2*pi, siendo pi=3.1416 declarada como constante), daría lugar a un error de compilación; lo que le confiere mayor seguridad a la hora de programar. **La inicialización del valor de una constante es obligatoria**.

Por otro lado, **las variables pueden cambiar de valor a lo largo de la ejecución del código**, de ahí su nombre. Su inicialización no es obligatoria, pero si se ejecutara una instrucción en donde se necesitara su valor preciso y esta no está o bien inicializada o bien asignada (p.ej. declarar una variable x y se ejecuta la instrucción y := x + 1 sin conferir anteriormente un valor a x), se producirá un error de compilación.

Evidentemente, también dará un error de compilación si ejecutamos una instrucción que utiliza una variable no declarada.

Por lo que, siguiendo las reglas de Pascal, las declaraciones se realizan antes del algoritmo, y las constantes y las variables se declaran separadas en dos secciones, una sucedida por la palabra reservada const y otra por la palabra var, siguiendo la siguiente sintaxis:

<declaración_constantes> ::= <nombre_constante> : <tipo_dato> = <valor_constante> ;

<declaración_variables> ::= <nombre_variable> : <tipo_dato> [= <valor_inicio>] ;

```
const
   PI : Real = 3.1416;
var
   A, r: Real;
```

Cabe destacar que **por convenio, las constantes se escriben en mayúsculas mientras que las variables se escriben en minúsculas**. Este convenio se establece porque en lenguajes de bajo nivel, como son débilmente tipados y por ende no se realizan declaraciones, los desarrolladores distinguen así entre lo que está diseñado para que sea constante y lo que está diseñado para que sea variable. Sin embargo, los lenguajes de bajo nivel como Python no logran evitar que a lo largo del programa se pueda modificar el valor de una constante, ya que son tratadas como variables igualmente.

Por otra parte, otros lenguajes como C no distinguen entre la parte donde se escriben las declaraciones de la parte donde se escriben las asignaciones y resto de instrucciones.

Aparte de constantes y variables, todo tipo de dato no primitivo del lenguaje también debe declararse. Dependiendo del tipo de dato, la manera de declararlos también variará, pues no es lo mismo una enumeración que una rodaja, por ejemplo.

En Pascal se procedería siguiendo esta sintaxis, sucedida de la palabra reservada type:

<declaración_enumeración> ::= <nombre_tipo> = (<tag1>, … , <tagn>);

<declaración_rodaja> ::= <nombre_tipo> = <valor_inicio>..<valor_fin>;

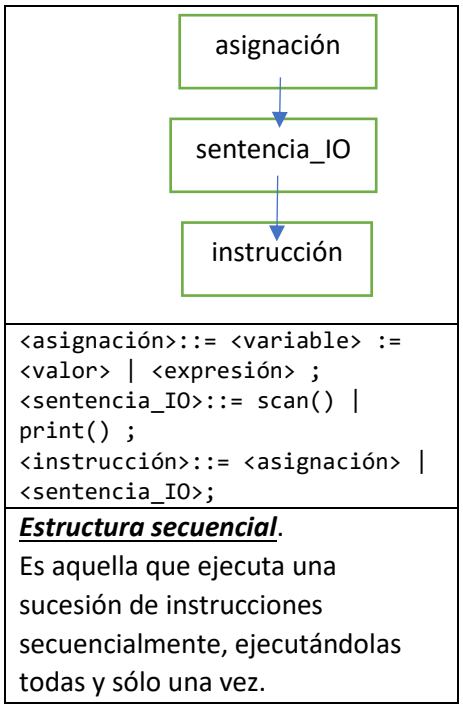
Así por ejemplo, para definir en un programa de Pascal la enumeración de días de la semana y la rodaja de nº positivos, se procedería de la siguiente manera:

```
type
   dias = (L, M, X, J, V, S, D);
   natural = 0..MaxInt;
```

Como siempre, esto depende bastante del lenguaje de programación, y de hecho, en C no se aceptan rodajas y en Python no se aceptan ninguno de los dos TDDs vistos por ahora.

4.2 Estructura secuencial

La **estructura secuencial** se refiere a un diseño del algoritmo en el cual **una serie de instrucciones son ejecutadas en el orden en el que aparecen**, de arriba a abajo.




```
<asignación>::= <variable> :=
<valor> | <expresión> ;
<sentencia_IO>::= scan() |
print() ;
<instrucción>::= <asignación> |
<sentencia_IO>;
```

Estructura secuencial.
Es aquella que ejecuta una sucesión de instrucciones secuencialmente, ejecutándolas todas y sólo una vez.

La estructura secuencial tiene las siguientes características:

- Sus instrucciones se siguen de forma que **el valor de salida que se obtiene tras ejecutar una instrucción es utilizado como valor de entrada para ejecutar su siguiente** instrucción
- El programa **sólo** tiene **un punto de entrada y sólo un punto de salida**
- La **ejecución** de las instrucciones es **unidireccional** (de arriba abajo, sin retroceder nunca), **lineal** (una a una y en orden) **y de una única vez**.
- La lectura de las instrucciones es sencilla, y por lo tanto **fácil de mantener**, ya que sólo se emplean **asignaciones y operaciones básicas** (entre ellas operaciones características de cada tipo de dato y operaciones de E/S que veremos en el tema 7)

En el caso del programa del área del círculo, observamos que se trata de un programa que se puede resolver mediante únicamente asignaciones, que actúa como función matemática (y por lo tanto, un único punto de entrada y un único punto de salida), que sólo es necesario ejecutar una sola vez un número de instrucciones, que tienen un orden concreto y que además, el resultado de la primera determina el resultado de la segunda. Por todo ello, podemos inferir que la estructura más adecuada para diseñar el algoritmo de este ejemplo es la estructura secuencial.

Siguiendo la sintaxis de Pascal de las asignaciones, que es la siguiente:

```
<asignación>::= <variable> := <valor> | <expresión> ;
```

Tendríamos únicamente una instrucción a ejecutar, ya que la instrucción de pseudocódigo en donde se había asignado a pi el valor 3.1416, se sustituye por la declaración de la constante. Esta instrucción corresponde con la ejecución de la fórmula matemática $A = \pi r^2$

```
A := PI*r*r;
```

Hay que tener en cuenta que la operación exponenciación (como la que se presenta, r^2), no existe en Pascal, por lo que debemos pensar en otras formas de ejecutar dicha operación. Sabemos que por definición: $x^n = x \cdot x \cdot x \cdot x \ldots$ n veces, luego, $x^2 = x \cdot x$.

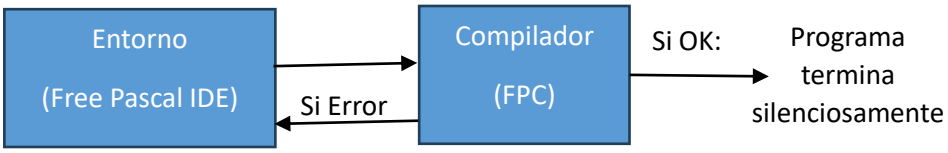
La secuencia de instrucciones de nuestro programa, en Pascal, debe ir comprendida entre las palabras reservadas begin y end. El programa en Pascal quedaría finalmente de la siguiente forma:

```
program area_circulo;
{$Assertions ON}
{ Datos de entrada: un real r, radio de la circunferencia
Precondición: r >0.0
Datos de salida: el área del círculo, A, nº real
Postcondición: r = PI * r^2, con PI = 3.1416}

const
   PI : Real = 3.1416;
var
   A, r: Real;
begin
   r := 1; {Inicializar datos de entrada}

   A := PI*r*r;

   Assert(A = PI); {Verificar caso de prueba}

end.
```

No obstante, a la hora de programar nuestros primeros programas, debemos tener en cuenta tres aspectos:

- Estos programas **no reciben entrada por teclado ni imprimen en pantalla**
- Por lo dicho anteriormente, esto causará un error de compilación porque los datos de entrada que no han sido inicializados tienen que procesarse en una asignación posterior. Por ello, es **obligatorio inicializar en el código los datos de entrada**, preferiblemente con los valores de un **caso de prueba** significativo
- No estaría de más **comprobar el correcto funcionamiento del programa**. Para ello, utilizamos la funcionalidad **Assert**. Para habilitarla, se ha de escribir tras la cabecera del programa `{$Assertions ON}`. La sintaxis de esta funcionalidad es: `<aserción>::= Assert( <dato_de_salida> = <valor_esperado> )` y con esto, **comprobamos que dados los datos de entrada del caso de prueba, obtengamos los datos de salida esperados**.

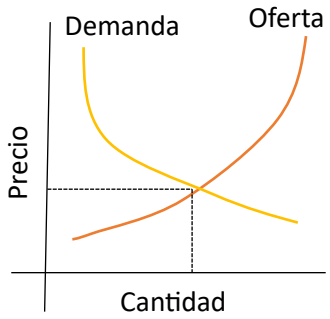
Si la aserción es verdadera, el programa termina silenciosamente, ya que ni recibe ni imprime datos en pantalla (no hay comunicación alguna con el usuario), lo que significa que **el programa es parcialmente correcto y** que **el caso de prueba se ha verificado** correctamente. **En caso contrario, el compilador lanzará un error de aserción** el cual será visible desde nuestro entorno de programación, por lo que **el programa no es correcto**.



A continuación, expondremos otro programa más complicado en el cual la serie de instrucciones a ejecutar ya no es tan obvia. Además, codificaremos este programa en ADA, C y Python para que se puedan observar las principales diferencias entre estos lenguajes y Pascal.

"Ecn. El principio de Economía que se utiliza en el ámbito empresarial para que las compañías que ofrezcan sus productos o servicios obtengan el máximo beneficio posible establece que lo ideal para el vendedor sería tener ventas altas y precios altos, pero esto contrasta con lo que el comprador desea, que son precios bajos. El principio que busca solución a este conflicto es la ley de la oferta y la demanda.

Si observamos el modelo de telaraña (gráfico a la derecha), se puede ver en un espacio donde sus variables son el precio del producto y la cantidad de productos existente dos funciones características, la de la oferta del vendedor (que aumenta el precio según la cantidad de productos a vender) y la de la demanda del comprador (que disminuye el precio según el mismo criterio)



El objetivo del ejercicio es encontrar el punto donde intersecan ambas funciones, el umbral de rentabilidad . Siendo la ecuación de la oferta $X_O = Q_O P - C_f$ y la ecuación de la demanda $X_D = Q_{max} - Q_D P$, siendo X_O, las unidades ofertadas; X_D, las unidades demandadas; C_f, los costes fijos de producción; Q_{max}, el stock máximo del producto; P, el precio de la unidad; y la pendiente de las rectas Q_O, Q_D.

Diseña un programa en el que dados los costes fijos de producción, el stock máximo del producto y las pendientes de ambas rectas, calcule el precio y la cantidad del producto necesarios para que estos valores cumplan con el umbral de rentabilidad"

En resumen, se pide el corte entre dos rectas de las que se saben sus ecuaciones y algunos parámetros, entre ellos C_f, Q_max y Q_O, Q_D; el corte entre ambas rectas es el punto (X, P), siendo X la cantidad y P el precio.

Si las rectas intersecan, los valores de X y P se igualan, por lo que X_O = X_D, por lo que esta igualdad $Q_O P - C_f = Q_{max} - Q_D P$, se cumple. De esta sencilla ecuación es donde despejaremos el valor de P, y una vez obtenido el precio de equilibrio, sustituimos en una de las dos rectas la incógnita P para obtener la cantidad de equilibrio X.

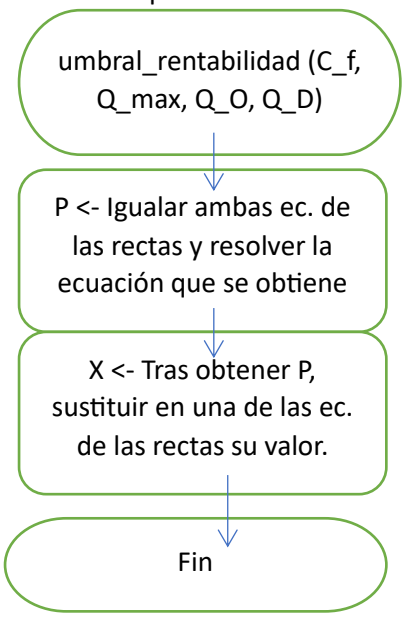
La solución que se obtiene al despejar P es $P = \frac{Q_{max}+C_f}{Q_O+Q_D}$. Tras obtener este valor, obtendremos X sustituyendo P por su valor p.ej. en la recta de la oferta.

La solución del problema en pseudocódigo sería la siguiente:

```
umbral_rentabilidad (C_f, Q_max, Q_O, Q_D; P, X){
      P := (Q_max + C_f) / (Q_O + Q_D);
      X := Q_O*P – C_f;
}
```

Luego, la especificación pre-post de este programa es la siguiente. Para simplificar el programa, asumiremos que todos los parámetros del programa son reales, aunque parámetros como cantidad de productos puedan optimizarse mejor como enteros:

- Datos de entrada: cuatro reales, C_f, Q_max, Q_O, Q_D; costes fijos, stock máximo y pendientes de las rectas de oferta y demanda.



- Precondición: C_f, Q_max, Q_O, Q_D >= 0.0
- Datos de salida: dos reales, P, X, precio y cantidad del umbral de rentabilidad
- Postcondición: P = (Q_max + C_f) / (Q_O + Q_D) ^ X = Q_O*P – C_f

Ya se ha resuelto prácticamente la mitad del problema, una vez se haya leído el enunciado e identificado lo que se pide, lo que se tiene, lo que se debe obtener y la serie de pasos a ejecutar, el último paso que se ejecuta es la implementación del algoritmo en el lenguaje de programación elegido, y verificar que el programa funciona correctamente.

Se muestra a continuación cómo se vería dicho programa en los siguientes lenguajes de programación:

1. **PASCAL**

```
program umbral_rentabilidad;
{$Assertions ON}
var
      C_f, Q_max, Q_O, Q_D, P, X: Real;
begin
      C_f := 500.0; Q_max := 700.0; Q_O := 3.0; Q_D := 1.0;
      P := (Q_max + C_f) / (Q_O + Q_D);
      X := Q_O*P – C_f;
      Assert(P = 300.0 and X = 400.0);
end.
```

2. **ADA**

```
with Ada.Assertions; use Ada.Assertions;
procedure umbral_rentabilidad (C_f, Q_max, Q_O, Q_D: in Float;
                               P, X: out Float) is
begin
      C_f := 500.0; Q_max := 700.0; Q_O := 3.0; Q_D := 1.0;
      P := (Q_max + C_f) / (Q_O + Q_D);
      X := Q_O*P – C_f;
      pragma Assert(P = 300.0 and X = 400.0);
end umbral_rentabilidad;
```

3. Python

```
# umbral_rentabilidad
C_f, Q_max, Q_O, Q_D = 500.0, 700.0, 3.0, 1.0
P = (Q_max + C_f) / (Q_O + Q_D)
X = Q_O*P – C_f
assert(P == 300.0 and X == 400.0)
```

4. C

```
#include <assert.h>
void main() { /* umbral_rentabilidad */
        double C_f, Q_max, Q_O, Q_D, P, X;
        C_f = 500.0; Q_max = 700.0; Q_O = 3.0; Q_D = 1.0;
        P = (Q_max + C_f) / (Q_O + Q_D);
        X = Q_O * P - C_f;
        assert(P == 300.0 && X == 400.0);
}
```

Las normas gramaticales que se han seguido para escribir el programa en cada lenguaje de programación se encuentran en el anexo relacionado a este tema. **Hay que tener en cuenta a la hora de programar las limitaciones de cada lenguaje y adaptar el pseudocódigo a estas**.

Tradicionalmente, debemos en primer lugar, declarar las variables de entrada, las de salida, y las auxiliares que se vayan a utilizar a lo largo de todo el programa, así como su tipo. Una vez declaradas, podemos operar con ellas, y exclusivamente con datos del tipo declarado.

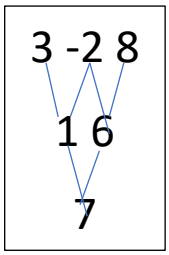
Por otra parte, C es un lenguaje bastante peculiar, pues no se distingue entre las partes donde se declaran variables de las que se asignan las instrucciones, y ambas se tratan como asignaciones por igual, salvo en la primera aparición de la variable, en donde sí es obligatorio por cada una de ellas declarar el tipo de dato correspondiente a la variable. Sólo se declaran variables en su primera aparición, si se realiza una vez declarada por primera vez, surgirá un error de compilación, al igual que asignar un valor a una variable cuyo tipo no se ha declarado.

Python es un lenguaje de bajo nivel, por lo que no es necesario declarar ninguno de los parámetros como reales u otro tipo de dato, directamente pasamos a las asignaciones. Esto quiere decir que si introducimos enteros, entonces el resultado puede salir o entero o real (p.ej. división no exacta entre dos enteros). Python tiene una manera muy especial de manejar los errores de tipado, y se verá en el tema 7, cuando veamos “Conversión de tipos”

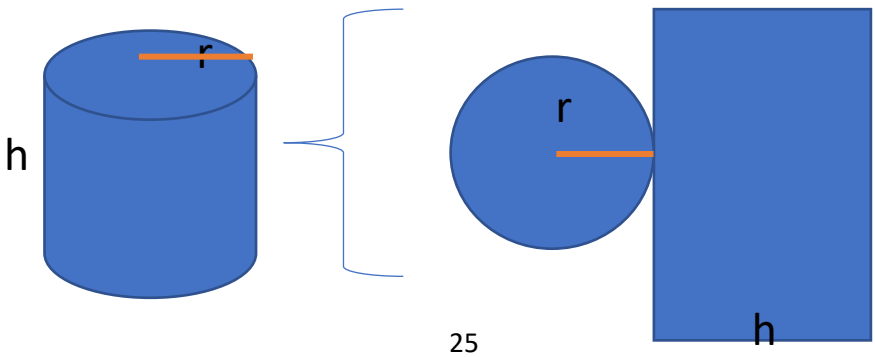
4.3 Ejercicios propuestos

Se sugieren realizar los siguientes ejercicios de programación. Estos ejercicios se deben realizar usando exclusivamente las herramientas que hemos visto hasta ahora, es decir, declaraciones y asignaciones. Se pueden hacer en el lenguaje de programación que uno crea conveniente, aunque preferiblemente usaremos Pascal; salvo que el enunciado indique que hay que escribir el programa en un lenguaje determinado. Los programas que denotan aplicaciones en el campo STEM se denotarán con la abreviatura de la rama STEM a la que pertenece.

65. (*) Mat. Diseña un programa en el que dados la pendiente de una recta m y su ordenada en el origen n, calcule el valor de y en el valor x del eje de abscisas. Recuerda que la ecuación de la recta es $y = mx + n$
66. (*) Diseña un programa en el que dadas diez notas medias de las diez asignaturas del curso, calcule la nota media del expediente académico en ese curso. Recuerda que para realizar la media se sumaban todos sus datos y se dividía dicha suma por el número de datos existente.
67. (*) Diseña un programa en el que dadas tres notas medias de un trimestre, calcula la nota media de la asignatura en ese año. Sin embargo, no todos los exámenes tienen el mismo peso, la evaluación del primer trimestre tiene un peso del 20% de la nota total, la del segundo trimestre, del 30% y la del tercero, el resto del peso.
68. (*) Diseña un programa en el que dados dos números enteros, indique si el valor de su suma devuelve un número par o si por el contrario, número impar
69. (*) Diseña un programa en el que dados dos números enteros, indique si el valor de su suma devuelve un número par o no pero esta vez sin realizar la suma
70. (*) Diseña un programa en el que dado un natural n, decide si este es o no un múltiplo satánico. n es múltiplo satánico cuando es múltiplo de 6 y de 66 o lo es de 666.
71. (*) En el videojuego “Brain Training”, uno de sus ejercicios de agilidad era “Triángulo Aritmético” En este minijuego se mostraban tres números enteros, y su objetivo era hallar lo más rápido posible la suma de los dos pares que forman dicha secuencia. P.ej. para la secuencia 3, -2, 8, la suma del primer par es 3 + -2 = 1, la suma del segundo par es -2 + 8 = 6, por lo que la suma de los dos pares es 1 + 6 = 7. Diseña el programa que calcule el triángulo aritmético de tres números dados



72. (**) Mat. Diseña un programa que calcule dado el radio r y su altura h el volumen de un cilindro. Recuerda que el volumen de un cuerpo geométrico es $V = A_{base} * h$, siendo el área de su base el área de un círculo ($A = \pi r^2$), pi lo aproximaremos a 3.1416
73. (**) Mat. Diseña un programa que calcule el área de un tanque de agua cilíndrico dado su radio r y su altura h. La base de arriba del cilindro queda descubierta, ya que desde allí se introduce el agua. Luego, necesitamos construir la base de abajo y la superficie cilíndrica que encierra el agua. Recuerda que la longitud de la circunferencia es $2\pi r$



74. (**) Diseña un programa en el que dados tres números reales a, b, c, devuelva para cada una de esas variables cuál es el porcentaje que supone entre el total que forman la suma de ellas tres.
75. (**) Diseña un programa en el que dados dos números enteros a y b, intercambie sus valores. P.ej. si se pasan los valores 3, 8; se deben devolver los valores 8, 3.
76. (**) Dado un número entero de cuatro cifras, diseña un programa que sea capaz de descomponerlo en sus cuatro dígitos, almacenando cada uno en una variable del tipo entero.
77. (**) Dado un carácter, diseña un programa que indique si este forma parte del alfabeto inglés o no. Así p.ej. la é, la ñ o la ü no entrarían en la definición.
78. (**) Repite el mismo programa anterior sin utilizar la función nativa isalpha()
79. (**) Dados dos caracteres x e y, siendo el primero mayúsculo y el segundo minúsculo, diseña un programa en el cual sus valores se intercambian. El programa debe respetar el hecho de que el carácter x sea siempre mayúsculo y el carácter y sea siempre minúsculo. P.ej. si se pasan los valores 'A', 'b'; se deben devolver los valores 'B', 'a'
80. (***) Dados dos dígitos a y b, diseña un programa que realiza la suma de sus dígitos. Esta se realiza sumando los valores numéricos de los dígitos y devolviendo el valor de dígito del resultado. P.ej. '3' + '2' = '5'. Pero a veces puede darse desborde, por ejemplo, en '6' + '9', sabemos que su suma da 15, pero devolverá su último dígito, '5', por lo que '6' + '9' = '5' El programa debe indicar también si hay desborde o no.
81. (***) Dadas dos letras minúsculas x e y, diseña un programa que indique si ambas letras forman un fonema trabado, es decir, si al juntar la letra x seguida de la letra y, se forman alguna de estas secuencias de consonantes: bl, br, cl, cr, dr, fl, fr, gl, gr, pl, pr, tr.
82. (***) Mat. Dados tres coeficientes a, b y c que forman parte de la ecuación cuadrática $ax^2 + bx + c = 0$, diseña un programa que devuelva sus dos soluciones reales. La operación raíz cuadrada debe implementarse aplicando el siguiente polinomio de aproximación, Δ (delta) denota el número del que se quiere calcular su raíz cuadrada:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\sqrt{\Delta} = 1 + \frac{1}{2}(\Delta - 1) - \frac{1}{8}(\Delta - 1)^2$$

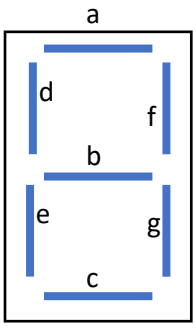
83. (***) Mat. Dados dos valores a, b del número complejo $a + bi$, diseña un programa que transforme la forma binómica del número en su forma polar $re^{i\theta}$, obteniendo los valores r y arg(θ). Recuerda que $r = \sqrt{a^2 + b^2}$ y que arg = $\arctan(\frac{b}{a})$, ángulo expresado en radianes y comprendido entre 0 y 2π. El polinomio de aproximación de la función arco tangente es:

$$\arctan(\theta) = \theta + \frac{\theta^3}{3}$$

84. (***) Fis. Diseña un programa que calcule el tiempo en segundos necesario para que un vehículo que vaya a v m/s pueda incorporarse en una autovía y alcanzar los vf m/s en el final de los x metros del carril de aceleración. v denota la velocidad inicial del vehículo al entrar al carril de aceleración y vf, la velocidad máxima de la autovía. Las ecuaciones del MRUA son:

$$MRUA\begin{cases} v = v_o + at \\ x = x_o + v_o t + \frac{1}{2}at^2 \\ v^2 = {v_o}^2 + 2ax \end{cases}$$

85. (***) Fis. Cuando un vehículo circula por una cuesta abajo, existe la posibilidad de que los frenos fallen, por ello, se diseñan apartaderos de frenado de emergencia. Diseña un programa en el que dada la velocidad máxima de una vía v en m/s, calcule la longitud necesaria para un apartadero diseñado para que el vehículo sea inmovilizado en t segundos. El vehículo entra a una velocidad máxima de un 30% superior a la dada
86. (****) Diseña un programa en el que dadas dos horas h1:min1:sg1 y h2:min2:sg2, calcule cuánto tiempo ha transcurrido desde la hora h1:min1:sg1 hasta la hora h2:min2:sg2, manteniendo en el resultado el formato de salida en horas, minutos y segundos. Estrategia: calcula para ambas horas cuántos segundos han transcurrido desde las 00:00 y luego convierte el resultado de su diferencia en horas, minutos y segundos.
87. (****) Repite el mismo programa que el anterior pero restando directamente ambas horas en su respectivo sistema sexagesimal #
88. (****) Inf. Este es un problema muy típico de la asignatura Diseño de Sistemas Digitales, que se imparte en el grado de Ingeniería Informática. El objetivo de este problema consiste en diseñar un decodificador de 7 segmentos, también llamado BCD (Binary Coded Decimal)



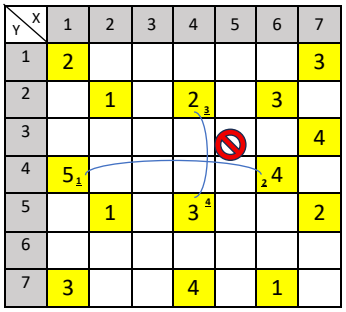
Este decodificador lo que hace es interpretar una serie de señales binarias, que corresponden al valor numérico en base 2 del dígito solicitado, y tras pasar dichos valores al decodificador, se devuelve como resultado una serie de siete señales binarias en forma de segmentos.

Aquí a la izquierda se muestra la idea de cómo se debería ver el número en pantalla. Este formato de número se suele encontrar en antiguas máquinas expendedoras o en termómetros públicos.

Así por ejemplo, si queremos imprimir en la pantalla el número 0, se encenderán todos los segmentos excepto el b, y para imprimir el 1, se encenderán únicamente los segmentos f, g.

Diseña un programa en el que dado un dígito, se configuran las expresiones lógicas por las que cada uno de los siete segmentos se enciende. Se pide también que la resolución del problema sea lo más efectiva posible, es decir si sabemos que un segmento se enciende o no para ciertos valores y estos se pueden agrupar en una característica común, expresarlo como subconjunto.

89. (****) Dados dos puentes, uno de extremos x1, y1; x2, y2 y otro de extremos x3, y3; x4, y4. (xi, yi simbolizan las coordenadas de la isla i del tablero Hashi), determina si dichos puentes pueden colocarse en el tablero simultáneamente sin superponerse uno sobre el otro. Los puentes dados descartan cualquier escenario en el que un puente pase por encima de una isla. Además, solo pueden disponerse de manera vertical (de arriba abajo) u horizontal (de izquierda derecha, en esos órdenes). #



90. (*****).Diseña un programa en el que dados una hora completa en horas, minutos y segundos, un día, un mes y un año, devuelva su hora epoch. La hora epoch se define como el tiempo transcurrido en segundos entre el 1 de enero de 1970 a las 00:00 y la fecha dada. Recuerda que no todos los meses tienen la misma cantidad de días: enero, marzo, mayo, julio, agosto, octubre y diciembre tienen 31 días; abril, junio, septiembre y noviembre tienen 30 días; y febrero tiene 28 días (29 días si es año bisiesto) Un año es bisiesto si este es múltiplo de 4 pero no un año secular (aquel que termina por 00) o bien este es múltiplo de 400 #

Los booleanos toman dos valores: 0 (false), 1 (true). Sin embargo, en Pascal, los booleanos devuelven expresamente las palabras false, true. Para indicarle al compilador que debe tratar los booleanos como números y evitar así un error de tipado, hay que escribir tras la cabecera del programa la regla {$mode c}

Tratar los booleanos como su concepción natural, apagado y encendido, es muy común en la formulación científica, ya que nos permite escribir cualquier expresión lógico-matemática como el producto de lo que hay que ejecutar y su función de activación. Si la función de activación dice “apagado”, entonces lo ejecutable también “se apaga”, se omite en la fórmula. P.ej. la expresión “10€ la entrada, 6€ niños y jubilados” puede escribirse como “10 – 4 * (edad <= 12 or edad >= 65)”. De este modo, -4 es nuestro ejecutable, el cual se activa si la edad del usuario es menor o igual que 12 o mayor o igual que 65. Si la edad está en ese rango de valores, -4 “se activa” en la expresión, y por lo tanto, se resta a 10 la cantidad 4 para obtener el precio de la entrada para niños y jubilados. En caso contrario, el -4 “se apaga”, quedando solo el 10.

NOTA: A la hora de realizar **aserciones con variables reales**, es posible que, debido a la aritmética de coma flotante, aunque teóricamente dicha aserción es correcta, por la aritmética del ordenador, se dé lugar a un resultado muy aproximado al esperado, y por tanto `assert(var_real = valor_esperado)` propague un error de aserción. Para paliar este efecto, se sugiere usar `assert(abs(var_real – valor_esperado) < 0.001)`