telecomunicaciones AT&T, se conoce popularmente por K&R C y está prácticamente en desuso. En los años 80, C fue modificado y estandarizado por el American National Standards Institute (ANSI), que dio lugar al denominado ANSI C y que ahora se conoce como C89 por el año en que se publicó. El estándar se revisó en los años 90 y se incorporaron nuevas características que mejoran sensiblemente el lenguaje. El resultado es la segunda edición del ANSI C, más conocida como C99. Esta es la versión que estudiaremos en este curso.

En la asignatura utilizaremos un compilador de C gratuito: el `gcc` en su versión 3.2 o superior. Inicialmente se denominó a `gcc` así tomando las siglas de GNU C Compiler. GNU es el nombre de un proyecto que tiene por objeto ofrecer un sistema operativo «libre» y todas las herramientas que es corriente encontrar en una plataforma Unix. Hoy día se ha popularizado enormemente gracias a la plataforma GNU/Linux, que se compone de un núcleo de sistema operativo de la familia del Unix (Linux) y numerosas herramientas desarrolladas como parte del proyecto GNU, entre ellas `gcc`. La versión de `gcc` que usaremos no soporta aún todas las características de C99, pero sí las que aprenderemos en el curso.

Cualquier distribución reciente de Linux[5] incorpora la versión de `gcc` que utilizaremos o una superior. Puedes descargar una versión de `gcc` y las utilidades asociadas para Microsoft Windows en `http://www.delorie.com/djgpp`. En la página `http://www.bloodshed.net/devcpp.html` encontrarás un entorno integrado (editor de texto, depurador de código, compilador, etc.) que también usa el compilador `gcc`.

¡Ojo!, no todos los compiladores soportan algunas características de la última versión de C, así que es posible que experimentes algún problema de compatibilidad si utilizas un compilador diferente del que te recomendamos.

1.4. Más allá de los programas: algoritmos

Dos programas que resuelven el mismo problema expresados en el mismo o en diferentes lenguajes de programación pero que siguen, en lo fundamental, el mismo procedimiento, son dos *implementaciones* del mismo *algoritmo*. Un algoritmo es, sencillamente, una secuencia de pasos orientada a la consecución de un objetivo.

Cuando diseñamos un algoritmo podemos expresarlo en uno cualquiera de los numerosos lenguajes de programación de propósito general existentes. Sin embargo, ello resulta poco adecuado:

- no todos los programadores conocen todos los lenguajes y no hay consenso acerca de cuál es el más adecuado para expresar las soluciones a los diferentes problemas,
- cualquiera de los lenguajes de programación presenta particularidades que pueden interferir en una expresión clara y concisa de la solución a un problema.

Podemos expresar los algoritmos en lenguaje natural, pues el objetivo es comunicar un procedimiento resolutivo a otras personas y, eventualmente, traducirlos a algún lenguaje de programación. Si, por ejemplo, deseamos calcular la media de tres números leídos de teclado podemos seguir este algoritmo:

1. solicitar el valor del primer número,
2. solicitar el valor del segundo número,
3. solicitar el valor del tercer número,

[5] En el momento en que se redactó este texto, las distribuciones más populares y recientes de Linux eran SuSE 8.2, RedHat 9, Mandrake 9.1 y Debian Woody.

La torre de Babel

Hemos dicho que los lenguajes de programación de alto nivel pretendían, entre otros objetivos, paliar el problema de que cada ordenador utilice su propio código de máquina. Puede que, en consecuencia, estés sorprendido por el número de lenguajes de programación citados. Pues los que hemos citado son unos pocos de los más utilizados: ¡hay centenares! ¿Por qué tantos?

El primer lenguaje de programación de alto nivel fue Fortran, que se diseñó en los primeros años 50 (y aún se utiliza hoy día, aunque en versiones evolucionadas). Fortran se diseñó con el propósito de traducir fórmulas matemáticas a código de máquina (de hecho, su nombre proviene de «FORmula TRANslator», es decir, «traductor de fórmulas»). Pronto se diseñaron otros lenguajes de programación con propósitos específicos: Cobol (Common Business Oriented Language), Lisp (List Processing language), etc. Cada uno de estos lenguajes hacía fácil la escritura de programas para solucionar problemas de ámbitos particulares: Cobol para problemas de gestión empresarial, Lisp para ciertos problemas de Inteligencia Artificial, etc. Hubo también esfuerzos para diseñar lenguajes de «propósito general», es decir, aplicables a cualquier dominio, como Algol 60 (Algorithmic Language). En la década de los 60 hicieron su aparición nuevos lenguajes de programación (Algol 68, Pascal, Simula 67, Snobol 4, etc.), pero quizá lo más notable de esta década fue que se sentaron las bases teóricas del diseño de compiladores e intérpretes. Cuando la tecnología para el diseño de estas herramientas se hizo accesible a más y más programadores hubo un auténtico estallido en el número de lenguajes de programación. Ya en 1969 se habían diseñado unos 120 lenguajes de programación y se habían implementado compiladores o intérpretes para cada uno de ellos.

La existencia de tantísimos lenguajes de programación creó una situación similar a la de la torre de Babel: cada laboratorio o departamento informático usaba un lenguaje de programación y no había forma de intercambiar programas.

Con los años se ha ido produciendo una selección de aquellos lenguajes de programación más adecuados para cada tipo de tarea y se han diseñado muchos otros que sintetizan lo aprendido de lenguajes anteriores. Los más utilizados hoy día son C, C++ , Java, Python, Perl y PHP.

Si tienes curiosidad, puedes ver ejemplos del programa «Hello, world!» en más de 100 de lenguajes de programación diferentes (y más de 400 dialectos) visitando la página http://www.uni-karlsruhe.de/~uu9r/lang/html/lang-all.en.html

4. sumar los tres números y dividir el resultado por 3,
5. mostrar el resultado.

Como puedes ver, esta secuencia de operaciones define exactamente el proceso que nos permite efectuar el cálculo propuesto y que ya hemos implementado como programas en Python y en C.

Los algoritmos son independientes del lenguaje de programación. Describen un procedimiento que puedes implementar en cualquier lenguaje de programación de propósito general o, incluso, que puedes ejecutar a mano con lápiz, papel y, quizá, la ayuda de una calculadora.

¡Ojo! No es cierto que cualquier procedimiento descrito paso a paso pueda considerarse un algoritmo. Un algoritmo debe satisfacer ciertas condiciones. Una analogía con recetas de cocina (procedimientos para preparar platos) te ayudará a entender dichas restricciones.

Estudia esta primera receta:

1. poner aceite en una sartén,
2. encender el fuego,
3. calentar el aceite,

4. coger un huevo,
5. romper la cáscara,
6. verter el contenido del huevo en la sartén,
7. aderezar con sal,
8. esperar a que tenga buen aspecto.

En principio ya está: con la receta, sus ingredientes y los útiles necesarios somos capaces de cocinar un plato. Bueno, no del todo cierto, pues hay unas cuantas cuestiones que no quedan del todo claras en nuestra receta:

- ¿Qué tipo de huevo utilizamos?: ¿un huevo de gallina?, ¿un huevo de rana?
- ¿Cuánta sal utilizamos?: ¿una pizca?, ¿un kilo?
- ¿Cuánto aceite hemos de verter en la sartén?: ¿un centímetro cúbico?, ¿un litro?
- ¿Cuál es el resultado del proceso?, ¿la sartén con el huevo cocinado y el aceite?

En una receta de cocina hemos de dejar bien claro con qué ingredientes contamos y cuál es el resultado final. En un algoritmo hemos de precisar cuáles son los datos del problema (datos de entrada) y qué resultado vamos a producir (datos de salida).

Esta nueva receta corrige esos fallos:

- Ingredientes: 10 cc. de aceite de oliva, una gallina y una pizca de sal.
- Método:
    1. *esperar a que la gallina ponga un huevo*,
    2. poner aceite en una sartén,
    3. encender el fuego,
    4. calentar el aceite,
    5. coger el huevo,
    6. romper la cáscara,
    7. verter el contenido del huevo en la sartén,
    8. aderezar con sal,
    9. esperar a que tenga buen aspecto.
- Presentación: depositar el huevo frito, sin aceite, en un plato.

Pero la receta aún no está bien del todo. Hay ciertas indefiniciones en la receta:

1. ¿Cuán caliente ha de estar el aceite en el momento de verter el huevo?, ¿humeando?, ¿ardiendo?
2. ¿Cuánto hay que esperar?, ¿un segundo?, ¿hasta que el huevo esté ennegrecido?
3. Y aún peor, *¿estamos seguros de que la gallina pondrá un huevo?* Podría ocurrir que la gallina no pusiera huevo alguno.

Para que la receta esté completa, deberíamos especificar con *absoluta precisión* cada uno de los pasos que conducen a la realización del objetivo y, además, cada uno de ellos debería ser realizable en *tiempo finito*.

No basta con decir *más o menos* cómo alcanzar el objetivo: hay que decir *exactamente* cómo se debe ejecutar cada paso y, además, cada paso debe ser realizable en tiempo finito. Esta nueva receta corrige algunos de los problemas de la anterior, pero presenta otros de distinta naturaleza:

- Ingredientes: 10 cc. de aceite de oliva, un huevo de gallina y una pizca de sal.
- Método:
    1. poner aceite en una sartén,
    2. encender el fuego a medio gas,
    3. calentar el aceite hasta que humee ligeramente,
    4. coger un huevo,
    5. romper la cáscara *con el poder de la mente, sin tocar el huevo*,
    6. verter el contenido del huevo en la sartén,
    7. aderezar con sal,
    8. esperar a que tenga buen aspecto.
- Presentación: depositar el huevo frito, sin aceite, en un plato.

El quinto paso no es *factible*. Para romper un huevo has de utilizar algo más que «el poder de la mente». En todo algoritmo debes utilizar únicamente instrucciones que pueden llevarse a cabo.

He aquí una receta en la que todos los pasos son realizables:

- Ingredientes: 10 cc. de aceite de oliva, un huevo de gallina y una pizca de sal.
- Método:
    1. poner aceite en una sartén,
    2. *sintonizar una emisora musical en la radio*,
    3. encender el fuego a medio gas,
    4. *echar una partida al solitario,*
    5. calentar el aceite hasta que humee ligeramente,
    6. coger un huevo,
    7. romper la cáscara,
    8. verter el contenido del huevo en la sartén,
    9. aderezar con sal,
    10. esperar a que tenga buen aspecto.
- Presentación: depositar el huevo frito, sin aceite, en un plato.

En esta nueva receta hay acciones que, aunque expresadas con suficiente precisión y siendo realizables, no hacen nada *útil* para alcanzar nuestro objetivo (sintonizar la radio y jugar a cartas). En un algoritmo, *cada paso dado debe conducir y acercarnos más a la consecución del objetivo*.

Hay una consideración adicional que hemos de hacer, aunque en principio parezca una obviedad: todo algoritmo bien construido debe finalizar tras la ejecución de un *número finito de pasos*.

Aunque todos los pasos sean de duración finita, una secuencia de instrucciones puede requerir tiempo infinito. Piensa en este método para hacerse millonario:

1. comprar un número de lotería válido para el próximo sorteo,
2. esperar al día de sorteo,
3. cotejar el número ganador con el nuestro,

4. si son diferentes, volver al paso 1; en caso contrario, somos millonarios.

Como ves, cada uno de los pasos del método requiere una cantidad finita de tiempo, pero no hay ninguna garantía de alcanzar el objetivo propuesto.

En adelante, no nos interesarán más las recetas de cocina ni los procedimientos para enriquecerse sin esfuerzo (¡al menos no como objeto de estudio de la asignatura!). Los algoritmos en los que estaremos interesados son aquellos que describen procedimientos de cálculo ejecutables en un ordenador. Ello limitará el ámbito de nuestro estudio a la manipulación y realización de cálculos sobre datos (numéricos, de texto, etc.).

Abu Ja'far Mohammed ibn Mûsâ Al-Khowârizm y Euclides

La palabra algoritmo tiene origen en el nombre de un matemático persa del siglo IX: Abu Ja'far Mohammed ibn Mûsâ Al-Khowârizm (que significa «Mohammed, padre de Ja'far, hijo de Moises, nacido en Khowârizm»). Al-Khowârizm escribió tratados de aritmética y álgebra. Gracias a los textos de Al-Khowârizm se introdujo el sistema de numeración hindú en el mundo árabe y, más tarde, en occidente.

En el siglo XIII se publicaron los libros *Carmen de Algorismo* (un tratado de aritmética ¡en verso!) y *Algorismus Vulgaris*, basados en parte en la *Aritmética* de Al-Khowârizm. Al-Khowârizm escribió también el libro «Kitab al jabr w'al-muqabala» («Reglas de restauración y reducción»), que dio origen a una palabra que ya conoces: «álgebra».

Abelardo de Bath, uno de los primeros traductores al latín de Al-Khowârizm, empezó un texto con «Dixit Algorismi...» («Dijo Algorismo...»), popularizando así el término *algorismo*, que pasó a significar «realización de cálculos con numerales hindo-arábigos». En la edad media los abaquistas calculaban con ábaco y los algorismistas con «algorismos».

En cualquier caso, el concepto de algoritmo es muy anterior a Al-Khowârizm. En el siglo III a.C., Euclides propuso en su tratado «Elementos» un método sistemático para el cálculo del Máximo Común Divisor (MCD) de dos números. El método, tal cual fue propuesto por Euclides, dice así: «Dados dos números naturales, a y b, comprobar si ambos son iguales. Si es así, a es el MCD. Si no, si a es mayor que b, restar a a el valor de b; pero si a es menor que b, restar a b el valor de a. Repetir el proceso con los nuevos valores de a y b». Este método se conoce como «algoritmo de Euclides», aunque es frecuente encontrar, bajo ese mismo nombre, un procedimiento alternativo y más eficiente: «Dados dos números naturales, a y b, comprobar si b es cero. Si es así, a es el MCD. Si no, calcular c, el resto de dividir a entre b. Sustituir a por b y b por c y repetir el proceso».

Un algoritmo debe poseer las siguientes características:

1. Ha de tener cero o más *datos de entrada*.
2. Debe proporcionar uno o más *datos de salida* como resultado.
3. Cada paso del algoritmo ha de *estar definido con exactitud*, sin la menor ambigüedad.
4. Ha de ser *finito*, es decir, debe finalizar tras la ejecución de un número finito de pasos, cada uno de los cuales ha de ser ejecutable en tiempo finito.
5. Debe ser *efectivo*, es decir, cada uno de sus pasos ha de poder ejecutarse en tiempo finito con unos recursos determinados (en nuestro caso, con los que proporciona un sistema computador).

Además, nos interesa que los algoritmos sean *eficientes*, esto es, que alcancen su objetivo lo más rápidamente posible y con el menor consumo de recursos.

EJERCICIOS

▶ **9** Diseña un algoritmo para calcular el área de un círculo dado su radio. (Recuerda que el área de un círculo es π veces el cuadrado del radio.)

▶ 10 Diseña un algoritmo que calcule el IVA (16%) de un producto dado su precio de venta sin IVA.

▶ 11 ¿Podemos llamar algoritmo a un procedimiento que escriba en una cinta de papel *todos* los números decimales de π?

. . . . . .