PROGRAMACIÓN (PRG). EUI. Curso 2001/2002

*T*ema 1. INTRODUCCIÓN

DSIC.- F. Marqués y N. Prieto

Índice General

1 Problemas, algoritmos y programas

Los conceptos que se desarrollarán a continuación son fundamentales en la mecanización del cálculo, objetivo de gran importancia en el desarrollo cultural humano pero que ha adquirido una relevancia extraordinaria con la aparición, y posterior universalización de los computadores. Problemas, algoritmos y programas forman el tronco principal en que se fundamentan los estudios de computación.

Dado un *problema* P, un *algoritmo* A es un conjunto de reglas, o instrucciones, que definen cómo resolver P en un tiempo finito.

Aunque "cambiar una rueda pinchada a un coche" es un problema que incluso puede estudiarse y resolverse en el ámbito informático, no es el tipo de problema que habitualmente se resuelve utilizando un computador. Por su misma estructura, y por las unidades de entrada/salida que utilizan, los ordenadores están especializados en el tratamiento de secuencias de información (codificada), como por ejemplo series de números, de caracteres, de puntos de una imagen, muestras de una señal, etc.

Ejemplos más habituales de las clases de problemas que se plantearán se pueden encontrar en el ámbito del cálculo numérico, del tratamiento de palabras, y de la representación gráfica. Así, algunos ejemplos de problemas son:

- Determinar el producto de dos números multidígito a y b
- Determinar la raíz cuadrada positiva del número 2

- Determinar la raíz cuadrada positiva de un número n cualquiera
- Determinar si el número n, entero mayor que uno, es primo
- Dada la lista de palabras l, determinar las palabras repetidas
- Determinar si la palabra p es del idioma castellano
- Separar silábicamente la palabra p
- Ordenar y listar alfabéticamente todas las palabras del castellano
- Dibujar en la pantalla del ordenador un círculo de radio r

Como se puede observar, en la mayoría de las ocasiones, los problemas se definen de forma general, haciendo uso de identificadores o *parámetros* (en los ejemplos esto es así excepto en el segundo problema, que es un caso particular del tercero). Las soluciones proporcionadas a esos problemas (algoritmos) tendrán también esa característica.

A veces los problemas están definidos de forma imprecisa puesto que los seres humanos podemos, o bien recabar nueva información sobre ellos, o bien realizar presunciones sobre los mismos. Cuando un problema se encuentra definido de forma imprecisa introduce una ambigüedad indeseable, por ello, siempre que esto ocurra, se deberá precisar el problema, eliminando en lo posible su ambigüedad. Así, por ejemplo, cuando en el problema tercero se desea determinar la raíz cuadrada positiva de un número n, se puede presuponer que dicho número n es real y no negativo, por ello redefinimos el problema del modo siguiente: determinar la raíz cuadrada positiva de un número n, entero no negativo, cualquiera.

Ejemplos de algoritmos pueden encontrarse en las secuencias de reglas aprendidas en nuestra niñez y mediante las cuales realizamos operaciones básicas de números multidígito, como por ejemplo sumas, restas, productos y divisiones. Son algoritmos ya que definen de forma precisa la resolución en tiempo finito de un problema de índole general. En lo siguiente se introducen otros ejemplos de algoritmos:

Ejemplo de algoritmo: considérese el problema enunciado en el listado anterior: ¿es n, entero mayor que uno, un número primo?

Como se recordará un número primo es aquel que sólo es divisible por el mismo o por la unidad. Los siguientes son posibles algoritmos:

```
Algoritmo 1.-
Considerar todos los números comprendidos entre 2 y n (excluido)
para cada número de dicha sucesión comprobar si ese
número divide al número n.
Si ningún número divide a n, entonces n es primo.
```

El algoritmo siguiente es similar al anterior, ya que la secuencia de cálculos que define para resolver el problema es idéntica a la definida por el algoritmo primero; sin embargo, se ha expresado utilizando una notación diferente.

```
Algoritmo 2.- Seguir los pasos siguientes en orden ascendente:
Paso 1 - Sea i un número entero de valor igual a 2.
Paso 2 - Si i es igual a n parar, n es primo
Paso 3 - Comprobar si i divide a n, entonces parar, n no es primo
Paso 4 - Reemplazar el valor de i por i+1, volver al Paso 2
```

El tercer algoritmo para resolver el problema mantiene una estrategia similar a la utilizada por los dos primeros: comprobaciones sucesivas de divisibilidad por números anteriores; sin embargo, haciendo uso de propiedades básicas de los números, mejora a los algoritmos anteriores al reducir mucho la cantidad de comprobaciones de divisibilidad efectuadas.

```
Algoritmo 3.- Seguir los pasos siguientes en orden ascendente:
Paso 1 - Si n vale 2 entonces parar, n es primo
Paso 2 - Si n es múltiplo de 2 (acaba en 2,4,6,8) acabar, n no es primo
Paso 3 - Sea i un número entero de valor igual a 3
Paso 4 - Si i es mayor que la raíz cuadrada positiva de n
         parar, n es primo
Paso 5 - Comprobar si i divide a n, entonces parar, n no es primo
Paso 6 - Reemplazar el valor de i por i+2, volver al Paso 4
```

Ejemplo de algoritmos: Palabras repetidas de la lista l

```
Algoritmo 1.-
- Ordenar la lista alfabéticamente,
- Recorrer la lista,
    si dos elementos consecutivos son iguales,
    entonces estaban repetidos, escribirlos

Algoritmo 2.-
- Recorrer la lista,
    para cada elemento comprobar (recorriendo de nuevo la lista)
    si está repetido y entonces escribirlo
```

Como es fácil ver, la descripción o nivel de detalle de la solución de un problema en términos algorítmicos depende de qué o quién debe entenderlo, resolverlo e interpretarlo.

Para facilitar la discusión se introduce el término genérico *procesador*. Se denomina *procesador* a cualquier entidad capaz de interpretar y ejecutar un cierto repertorio de instrucciones.

Un *programa* es un algoritmo escrito con una notación precisa para que pueda ser ejecutado por un procesador. Habitualmente, los procesadores que se utilizarán serán computadores con otros programas para facilitar el manejo de la máquina subyacente.

Cada instrucción al ejecutarse en el procesador supone cierto cambio o transformación, de duración finita, y de resultados definidos y predecibles. Dicho cambio se

produce en los valores de los elementos que manipula el programa. En un instante dado el conjunto de dichos valores se denomina el *estado del programa*.

Denominamos *cómputo* a la transformación de estado que tiene lugar al ejecutarse una o varias instrucciones de un programa.

2 Programas y la actividad de la programación

Como se ve, un programa es la definición precisa de una tarea de computación; siendo el propósito de un programa su ejecución en un procesador; y suponiendo dicha ejecución cierto cómputo o transformación.

Para poder escribir programas de forma precisa y no ambigua es necesario definir reglas que determinen tanto lo que se puede escribir en un programa (y el procesador podrá interpretar) como el resultado de la ejecución de dicho programa por el procesador. Dicha notación, conjunto de reglas y definiciones, es lo que se denomina un *lenguaje de programación*. Más adelante se estudiarán las características de algunos de ellos.

Como es lógico, el propósito principal de la programación consiste en describir la solución computacional (eficiente) de clases de problemas. Aunque hay que destacar que se ha demostrado la existencia de problemas para los que no puede existir solución computacional alguna, lo que implica una limitación importante a las posibilidades de la mecanización del cálculo.

Adicionalmente, los programas son objetos complejos que habitualmente necesitan modificaciones y adaptaciones. De esta complejidad es posible hacerse una idea si se piensa que algunos programas (la iniciativa de defensa estratégica de los EEUU, por ej.) pueden contener millones de lineas y que, por otro lado, un error en un único carácter de una sola linea puede suponer el malfuncionamiento de un programa (así, por ej. el Apollo XIII tuvo que cancelar, durante el trayecto, una misión a la luna debido a que en un programa se había sustituido erróneamente una coma por un punto decimal).

Debido a la complejidad mencionada, se considera que los programas tienen un ciclo de existencia que está formado, a grandes rasgos, por las dos etapas siguientes:

- Desarrollo: creación inicial y validación del programa
- Mantenimiento: correciones y cambios posteriores al desarrollo

También se puede establecer la siguiente subdivisión en función de la envergadura del problema a resolver (y del tamaño del programa necesario para resolverlo)

- Programación a pequeña escala: número reducido de lineas de programa, intervención de una sola persona, por ejemplo: un programa de ordenación.
- Programación a gran escala: muchas lineas de programa, equipo de programadores, por ejemplo: desarrollo de un sistema operativo

3 Lenguajes y modelos de programación

Los orígenes de los lenguajes de programación se encuentran en las máquinas. La llamada máquina original de Von Neumann se diseñó a finales de los años 40 en Princeton(aunque su diseño coincide en gran medida con el de la máquina de Charles Babbagge y Ada Byron de 1880). Los mayoría de los ordenadores modernos tienen tanto en común con la máquina original que se les denomina máquinas con arquitectura "Von Neumann".

La característica fundamental de dicha arquitectura es la banalización de la memoria, esto es, la existencia de un espacio de memoria único y direccionable individualmente, que sirve para mantener tanto datos como instrucciones. Existiendo unidades especializadas para el tratamiento de los datos (ALU) y de las instrucciones (U. Control). Esta es también, a grandes rasgos, la estructura de casi cualquier computador moderno.

Al nivel de la máquina, un programa es una sucesión de palabras (compuestas de bits), habitualmente en posiciones consecutivas de memoria que representan instrucciones o datos. El lenguaje con el que se expresa es el *lenguaje máquina*.

Por ejemplo, el fragmento siguiente (parte derecha) es una secuencia de código en lenguaje máquina

```
Load 24,         /* a está en la dir. 24h            10111100 00100100
Multiply 33,     /* mult. por b en la dir. 33h       10111111 00110011
Store 3C,        /* almacenar en c en la dir. 3Ch    11001110 00111100
```

Obviamente, los programas en lenguaje máquina son ininteligibles. Aunque no tanto, también son muy difíciles de entender los denominados *lenguajes ensambladores* (fragmento anterior, parte izquierda) en los que ya se utilizan mnemónicos e identificadores para las instrucciones y datos.

Estos lenguajes se conocen como de *bajo nivel*. Los problemas principales de dichos lenguajes son el bajo nivel de las operaciones que aportan, así como la posibilidad de efectuar todo tipo de operaciones (de entre las posibles) sobre los datos que manipulan. Así, por ejemplo, es habitual disponer tan solo de operaciones de caracter aritmético, de comparación y de desplazamiento, ello permite interpretar cualquier posición de memoria exclusivamente como un número. Un caracter se representará mediante un código numérico, aunque será visto a nivel máquina como un número (con lo que pueden multiplicarse entre si, por ej., dos caracteres, lo que posiblemente no tiene sentido).

En la década de los años 50 aparecieron lenguajes de programación dirigidos a hacer los programas más potentes, inteligibles y seguros. Un segmento como el anterior, para multiplicar ciertos valores a y b, dando como resultado c, podría ser simplemente:

```
c = a * b;
```

que, además de más legible, es más seguro puesto que implica que para poderse ejecutar, típicamente se comprueba que los datos implicados deben de ser numéricos. Por ejemplo, si a, b o c se hubiesen definido previamente como caracteres, la operación anterior puede no tener sentido, y el programa detenerse antes de su ejecución, advirtiendo de ello al programador, que podrá subsanar el error.

Así, por ejemplo, la motivación fundamental del primer lenguaje de alto nivel, el FORTRAN (FORmula TRANslator), desarrollado en 1957, era la de disponer de un lenguaje conciso para poder escribir programas de índole numérica y traducirlos automáticamente a lenguaje máquina.

Esta forma de trabajo es la utilizada hoy en día de forma habitual. A los programas que traducen las instrucciones de un lenguaje de alto nivel a un lenguaje máquina se les denomina compiladores, siendo el proceso seguido el siguiente:

Programa ⇒ Compilador(traducción) ⇒ Código máquina ⇒ Procesador(Ejecución)

Otros lenguajes de programación que aparecieron en la década de los 60, poco tiempo despues del FORTRAN son el Cobol, el LISP, el Basic y el PL1.

Algunas características comunes a todos ellos y, en general a todos los lenguajes de alto nivel son:

- Tienen operadores y estructuras cercanas a las utilizadas por las personas.
- Son más seguros que el código máquina y protegen de errores evidentes.
- El código que proporcionan es transportable (indep. de la máquina).
- El código que proporcionan es más legible.

En la década de los 70, como reacción a la falta de claridad y desestructuración introducida en los programas por los abusos que permitían los primeros lenguajes de programación, se originó la, así denominada *programación estructurada*, que consiste en el uso de un conjunto de modos de declaración y constructores en los lenguajes, reducido para que sea fácilmente abarcable y, al mismo tiempo, suficiente para expresar la solución algorítmica de cualquier problema resoluble.

Ejemplos de dichos lenguajes son los conocidos Pascal, C y Módula-2.

El modelo introducido por la *programación estructurada* tiene aún hoy en día una importancia extraordinaria para el desarrollo de programas. Este modelo será seguido implícita y explícitamente a lo largo de este curso.

Otro aspecto significativo de los lenguajes de programación de alto nivel que hay que destacar es el de que los mismos representan un procesador o *máquina extendida*: aquella que puede ejecutar las instrucciones de dicho lenguaje. Consideraremos que un lenguaje de programación es una extensión de la máquina en que se apoya, del mismo modo que un programa es una extensión del lenguaje de programación en que se construye.

Un lenguaje de programación proporciona un *modelo de computación* que no tiene por que ser igual al de la máquina que lo sustenta, pudiendo ser de hecho completamente diferente. Por ejemplo, un lenguaje puede hacer parecer que un programa se está ejecutando en varias máquinas distintas, aun cuando sólo existe una; o, por el contrario, puede hacer parecer que se está ejecutando en una sola máquina (muy rápidamente) cuando realmente ha subdividido la computación que representa entre varias máquinas.

A lo largo de la historia los seres humanos hemos desarrollado varios modelos de computación posibles (unos basados en una máquina universal, otros en las funciones

recursivas, otros en la noción de inferencia, etc). Se ha demostrado que todos estos modelos son computacionalmente equivalentes, esto es: si existe una solución algorítmica para un problema utilizando uno de los modelos, también existe una solución utilizando cualquiera de los otros.

El modelo más extendido de computación hace uso de una máquina universal bastante similar a la diseñada por Von Neumann. En este modelo una computación es una transformación de estados y un programa representa una sucesión de computaciones, o transformaciones, del estado inicial del problema al final o solución del mismo. Este modelo es el que seguiremos a lo largo del presente curso. En él, la solución de un problema se define dando una secuencia de pasos que indican la secuencia de computaciones para resolverlo. Este modelo de programación, que se denomina *modelo imperativo*, es el que se seguirá durante el presente curso.

Un diagrama de la evolución de algunos lenguajes puede encontrarse en: (Sethi)

4 La programación orientada a objetos. El lenguaje Java

La programación orientada a objetos es un nuevo modelo de programación que está adquiriendo preponderancia actualmente. Presenta mejoras para el desarrollo de programas en comparación a lo que aporta la programación estructurada que, como se ha mencionado, fue el modelo de desarrollo fundamental durante la década de los 70.

El elemento central de un programa orientado a objetos es la *clase*. Una clase determina completamente el comportamiento y las características propias de sus componentes. A los casos particulares de una clase se les denomina *objetos*. Un programa se entiende como un conjunto de objetos que interactuan entre si.

Una de las principales ventajas de la programación orientada a objetos es que facilita el reuso del código ya realizado, al tiempo que permite ocultar detalles no relevantes, aspectos fundamentales en la gestión de proyectos de programación complejos.

El lenguaje Java (1991) es un lenguaje orientado a objetos, de aparición relativamente reciente. En ese sentido, un programa en Java consta de una o más clases interdependientes. Las clases permiten describir las propiedades y habilidades de los objetos de la vida real con los que el programa tiene que tratar.

El lenguaje Java presenta, además, algunas características que lo diferencian, a veces significativamente, de otros lenguajes y en particular está diseñado para facilitar el trabajo en la WWW, mediante el uso de los programas navegadores de uso completamente difundido hoy en día. Los programas de Java que se ejecutan a través de la red se denominan *applets* (aplicación pequeña).

Otras de sus características son: la inclusión en el lenguaje de una herramienta para la programación gráfica y el hecho de que su ejecución es independiente de la plataforma, lo que significa que un mismo programa se ejecutará exactamente igual en diferentes sistemas.

Para la consecución de las características anteriores, el Java hace uso de lo que se denomina *Máquina Virtual Java* (Java Virtual Machine, JVM). La JVM es una extensión (mediante un programa) del sistema real en el que se trabaja, que permite

ejecutar el código resultante de un programa Java ya compilado independientemente de la plataforma en que se esté utilizando. En particular, todo navegador dispone de una JVM; de ahí la universalidad de su uso.

El procedimiento necesario para la ejecución un programa en Java es, de forma resumida, el siguiente:

Programa ⇒ compilador ⇒ Código JVM ⇒ JVM (Ejecutada en el sistema real)

5 Ejemplos

En los ejemplos siguientes se muestra, a título ilustrativo, el algoritmo, ya presentado en el punto primero del tema, para determinar si un número n entero y positivo es o no un número primo (Algoritmo 3). Dicho algoritmo se ha escrito utilizando distintos lenguajes de programación, todos ellos de uso muy difundido actualmente.

- Algoritmo en lenguaje Pascal:

```
function es_primo(n:integer):boolean;
(*determina si n, entero mayor que uno,  es un número primo*)
var i,integer; primo:boolean; raiz:real;
begin
  if n = 2 then primo:=true
  else if n mod 2 = 0 then primo:=false
  else begin
         primo:=true;
         i:=3; raiz:=sqrt(n);
         While (i<=raiz) and primo do
         begin
           primo:=((n mod i) <> 0);
           i :=i+2;
         end;
       end;
  es_primo:= primo;
end;
```

- Algoritmo en lenguaje C:

```
int es_primo(int n) {
/*determina si n, entero mayor que uno,  es un número primo*/
int i, primo; float raiz;

  if (n==2) primo = 0;
  else if (n%2) primo = 1;
  else {
```

```
        i = 3; raiz = sqrt(n);
        while ((i <= raiz) && !(n%i)) {i += 2;}
        primo = !(n%i);
  }
  return primo;
}
```

- Algoritmo en lenguaje Java:

```
static boolean es_primo(int n) {
// determina si n, entero mayor que uno, es un número primo
int i; double raíz; boolean primo;

  if (n==2) primo = true;
  else if (n%2=0) primo = false;
  else {
        i = 3; raíz = Math.sqrt(n);
        while ((i <= raíz) && (n%i != 0)) {i += 2;}
        primo = (n%i != 0);
  }
  return primo;
}
```