

El Retorno de

WALL·E



© 2008 WALT DISNEY PICTURES

Proyecto de Programación

Carrera de Ciencia de la Computación

Universidad de La Habana

2018-2019



INTRODUCCIÓN

Sobre un mundo rectangular dividido en casillas se ubica un conjunto de objetos. Algunos de estos objetos son robots que pueden ser programados para que efectúen tareas (incluso, competir entre ellos). Los robots pueden desplazarse hacia delante o hacia atrás, o girar 90 grados hacia la izquierda o hacia la derecha. La dirección del robot siempre será alguno de los ángulos (0° , 90° , 180° o 270° con respecto al norte). En el terreno puede haber objetos grandes (obstáculos), objetos

medianos (movibles) y objetos pequeños (cargables). Si un robot avanza sobre un obstáculo no se mueve, si avanza sobre un objeto movable (siempre que se pueda) lo desplaza y si avanza sobre un objeto pequeño lo recoge (lo almacena en su interior) o lo desplaza si tiene el interior ocupado. Si la casilla de enfrente al robot está libre el robot puede soltar (descargar) el objeto en su interior.

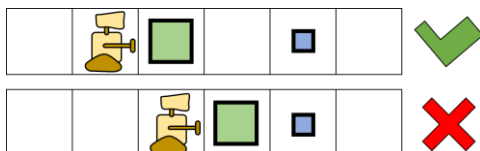
El robot tiene un núcleo central de procesamiento que es capaz de ejecutar un conjunto de instrucciones. Además tiene sensores que permiten “percibir” características del ambiente. Por ejemplo, un sensor ultrasónico para determinar distancia o una webcam para identificar el color del objeto enfrente más cercano. Los sensores permiten recibir información del medio y que el robot pueda tomar decisiones de acuerdo a ello.

OBJETOS

Los objetos pueden ser clasificarse según distintos atributos (tamaño, forma y color). El tamaño y la forma del objeto determinan si éste puede ser recogido, empujado o si es un obstáculo que impide el avance del robot. El tamaño del objeto frente al robot se puede conocer a través del atributo **size** y su forma a partir del atributo **shape**. Además los objetos tienen un color (accesible a través del atributo **color**) y un código de barra (atributo **number**). Estos últimos no influyen en la interacción entre el robot y los objetos pero pueden ser utilizados para especificar objetivos del robot (i.e. encontrar la bola negra 8).

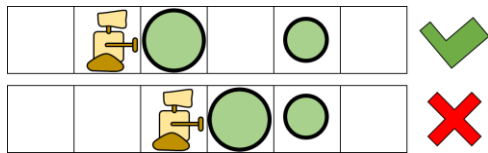
Si el objeto es pequeño (no importa la forma) el robot puede agarrarlo (moviéndose para la casilla donde se encuentra dicho objeto). Si el objeto es mediano (o pequeño y el robot ya contiene un objeto pequeño), este puede ser empujado. Las reglas para desplazar un objeto es la siguiente:

- Un robot nunca puede desplazar a otro robot.
- Una caja mediana o pequeña es desplazable siempre y cuando la siguiente casilla esté desocupada.

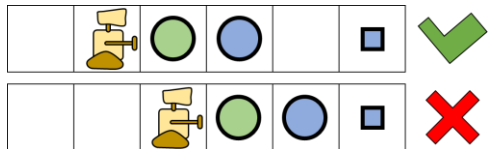


- Una bola grande es desplazable siempre y cuando la siguiente casilla esté desocupada.



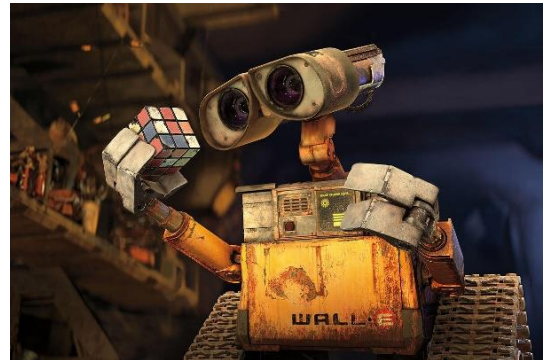


- Una bola mediana o pequeña es desplazable siempre y cuando luego de una serie de bolas pequeñas o medianas contiguas haya una casilla desocupada.



Si un robot suelta un objeto lo hace a la casilla del frente. Si ya existe un objeto en dicha casilla queda sin efecto la acción.

Para evaluar el comportamiento de los robots en un ambiente controlado se diseñó un lenguaje de programación denominado MATLAN. En este lenguaje las instrucciones aparecen en una matriz y la ejecución (a diferencia de instrucciones en secuencia) ocurre de forma lineal en cualquier dirección dentro de la matriz. Con el lenguaje se pueden declarar el programa de cada robot.



LENGUAJE MATLAN

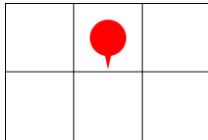
El lenguaje MATLAN (**Matrix Language**) es un lenguaje visual e imperativo con una memoria única compartida por todas las rutinas. En este se pueden declarar rutinas (método, procedimiento) o describir un conjunto de instrucciones a ser ejecutadas por el robot.

Toda rutina (incluida la rutina principal) está formada por una matriz bidimensional de instrucciones. La misma comienza ejecutando por una instrucción única “**start**” que debe estar ubicada en alguna casilla. La dirección de comienzo es siempre hacia abajo.



La “ejecución” recorre todas las casillas en determinada dirección y ejecuta cada instrucción que se encuentra a su paso. Si la ejecución se sale de los bordes de la matriz de instrucciones se dice que el programa (o rutina) ha finalizado. Si se está en el programa principal se termina la simulación, si se está en alguna subrutina se retorna al punto donde fue llamada y se continúa en la misma dirección que se tenía antes de hacer la llamada.

La siguiente figura representa un posible programa MATLAN que no hace nada.



Las casillas vacías no influyen en la ejecución del programa (son instrucciones que no hacen nada) ni cambian el flujo de la ejecución.






Si se desea terminar la ejecución de una rutina al llegar a una casilla se puede hacer explícitamente mediante la instrucción (**return**).








INSTRUCCIONES DEL ROBOT

La principal idea detrás de un programa es expresar la lógica de cómo funciona un robot. Qué debe hacer, cómo debe reaccionar ante el ambiente en que se encuentra. En el lenguaje existen dos tipos de instrucciones sobre los robots: acciones y sensores. Las acciones indican qué debe hacer el robot y los sensores permiten captar valores sobre las características presentes en los objetos que el robot tiene enfrente o alrededor.

La siguiente tabla describe las acciones principales que puede ejecutar un robot.

Instrucción	Nombre	Descripción
	Avanzar (<i>forward</i>)	El robot avanza una casilla siempre que no haya un obstáculo delante o no se salga de las dimensiones del terreno (imagine que los bordes están rodeados de bloques). Si se encuentra un objeto pequeño y el robot no está lleno, lo recoge. Si no, si se encuentra algún objeto intentará desplazarlo.
	Retroceder (<i>backward</i>)	El robot retrocede una casilla siempre que sea una casilla vacía. Hacia atrás no se recogen objetos ni se desplazan.
	Rotar derecha (<i>right</i>)	Rota el robot 90 grados hacia la derecha. Esta acción es siempre posible.
	Rotar izquierda (<i>left</i>)	Rota el robot 90 grados hacia la izquierda. Esta acción es siempre posible.
	Soltar (<i>drop</i>)	Expulsa el objeto que tiene el robot en su interior (si tiene alguno).

El siguiente programa ilustra cómo puede implementarse un robot que camina dos pasos, gira a la derecha y avanza nuevamente una casilla más (las casillas vacías de la derecha no tienen efecto pero ilustran que la matriz puede tener cualquier dimensión siempre y cuando se puedan disponer las instrucciones necesarias).

MEMORIA

Los robots tienen memoria para almacenar los programas, una **pila** con valores, un conjunto fijo de registros por cada llamada a subrutina y una memoria unidimensional que puede ser accedida para lectura y escritura en cualquier posición. La pila comienza la ejecución vacía y la memoria lineal comienza con todos los valores en 0 (La memoria lineal tiene una capacidad fija de 1000000 de enteros). Todos los registros tienen valor inicial 0 al inicio de la subrutina.

Las instrucciones en el programa pueden apilar valores, extraer valores o extraer, operar y apilar. A continuación se describen algunas instrucciones que acceden a la pila.

ALMACENANDO UNA CONSTANTE EN LA PILA

En MATLAB no se puede escribir valores numéricos. En su lugar existen tres instrucciones que permiten operar sobre la pila y “construir” cualquier valor entero a partir de su representación binaria. Las siguientes instrucciones están para ello (aunque pueden utilizarse para cualquier otra cosa que se necesite).

Instrucción	Nombre	Descripción
#	Número (number)	Apila el valor 0.
0	Cero (zero)	Extrae un valor de la pila y apila el doble de dicho valor.
1	Uno (one)	Extrae un valor de la pila y apila el doble de dicho valor incrementado en 1.

Note que la secuencia de instrucciones **#1011** permite poner en la pila el valor $11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$.

→ 0, **1** → $0 \cdot 2 + 1 = 1$, **0** → $1 \cdot 2 = 2$, **1** → $2 \cdot 2 + 1 = 5$, **1** → $5 \cdot 2 + 1 = 11$

OPERADORES

Los operadores en MATLAB pueden trabajar con 1, 2 o 3 valores de la pila. Siempre dejan en la pila el valor resultante de la operación. Las operaciones válidas en MATLAB son las mismas que en C#. El primer valor que se extrae de la pila es el último operando. A continuación se muestran algunos operadores y cómo pueden ser utilizados.

Sumar 5 + 3: **#101#11+**

Obtener -6: **##110-**

Los operadores condicionales y booleanos (&&, || y !) operan sobre enteros asumiendo que los valores distintos de 0 son **true** y el 0 es **false**. Un resultado **true** se expresa con 1 y un resultado **false** con 0.

Note que la siguiente secuencia de instrucciones deja 1 en la pila.

#	1	0	#	0	1	&&
---	---	---	---	---	---	----



A diferencia del operador lógico & que realiza el **and** bit a bit, por lo que dejaría 0 en la pila.

#	1	0	#	0	1	&
---	---	---	---	---	---	---

Operadores	Nombres	Cantidad de operandos
+, -, *, /, %	add, sub, mul, div, mod	2
++, --	inc, dec	1
&, , ^, ~	land, lor, lxor, lnot	2, 2, 2, 1
&&, , !	and, or, not	2, 2, 1
==, !=	eq, neq	2
<, <=, >, >=	lt, leq, gt, geq	2
?	ternary	3



REGISTROS

Los registros funcionan como variables locales en cada llamada a una subrutina. Estas pueden tomar valor de la pila o poner su valor en la pila. Los registros se indican con las letras de la A a la Z y para cada uno existen dos instrucciones.

Instrucción	Nombre	Descripción
	Poner A (getA)	Apila el valor de A.
	Salvar en A (setA)	Extrae un valor de la pila y lo asigna al registro A.




MEMORIA LINEAL





La memoria lineal funciona como un gran array que puede ser accedido para leer o almacenar un valor en cualquiera de sus posiciones. Este array es común para todas las rutinas de un robot. Para ello existen dos instrucciones.

Instrucción	Nombre	Descripción
	Leer del array (getAt)	Extrae un valor de la pila que indica el índice en el array que será consultado. Almacena en la pila el valor del array en dicha posición.
	Asignar en array (setAt)	Extrae dos valores de la pila. El primero indica el índice y el segundo el valor que será almacenado. Asigna dicho valor en la memoria lineal en la posición especificada por el índice.

SENSORES

Los sensores permiten al robot percibir características del entorno y apilan el valor de “lectura” cada vez que se ejecuta la instrucción.

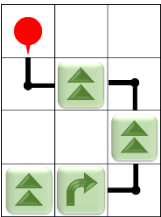
Instrucción	Sensor	Nombre	Valores y descripción
	Ultrasónico	distance	Entero indicando la cantidad de casillas desocupadas enfrente del robot (dentro de los límites del mundo).
	Webcam	color	Entero indicando el color del objeto enfrente. Si no hay objeto se apila 0.
	Kinect	shape	Entero indicando la forma del objeto enfrente. Si no hay objeto se apila 0.

	Barcode scanner	<code>code</code>	Entero indicando el código de barra del objeto enfrente. Si no hay objeto se apila 0.
	Pesa	<code>loaded</code>	1 si hay un objeto en el interior del robot, 0 en caso contrario.
	Cronómetro	<code>time</code>	Entero indicando el tiempo desde que comenzó la “vida” del robot. (Cantidad de rondas de la simulación). En cada ronda los robots ejecutan su programa hasta la siguiente “instrucción” de robot.
	Brújula	<code>direction</code>	Entero indicando la orientación del robot.

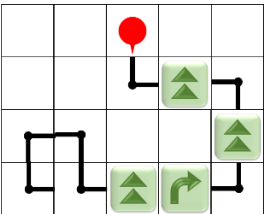
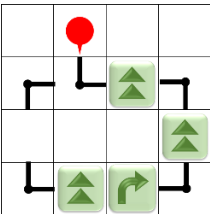
CONTROL DE FLUJO

La ejecución en MATLAN no es siempre secuencial. Algunas instrucciones permiten desviar el sentido de la ejecución e incluso terminarla.





Los angulares permiten cambiar la orientación del flujo para que no sea siempre en la misma dirección. El siguiente programa es válido y representa el mismo ejemplo del programa anterior.







Note que basándonos en angulares podemos lograr ciclos pero que serían infinitos. Incluso, repetir las instrucciones pero en sentido contrario.



Si se llega a la instrucción de inicio (`start`) se repite el proceso nuevamente siempre en dirección hacia abajo. Note que los angulares sólo afectan el sentido de la ejecución si ésta llega a la casilla por uno de los extremos del angular. Si se llega por alguno de los otros dos lados la ejecución mantiene su dirección.

Instrucción	Nombre
	<code>NE</code>
	<code>SE</code>
	<code>NW</code>
	<code>SW</code>

Las condicionales (**branch**) son instrucciones que funcionan de la siguiente forma. Extrae un valor de la pila. Si éste es falso desvía la ejecución hacia la izquierda (dependiendo de por donde se venga) y si es verdadero (distinto de 0) desvía la ejecución hacia la derecha.

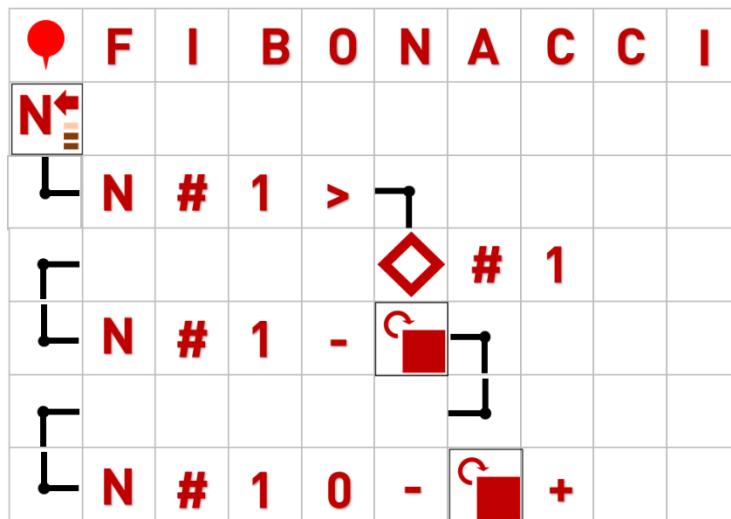
Instrucción	Nombre
	TS
	TE
	TN
	TW

El robot se carga con una serie de programas (indizados empezando en el índice 0 para el programa principal). Cada rutina tiene su propia matriz para la definición de sus instrucciones y su instrucción de inicio. Las rutinas pueden llamarse a sí mismas permitiendo la recursividad!

Para invocar una rutina se debe utilizar la instrucción `call`. Las siguientes instrucciones están relacionadas con la invocación a rutinas.

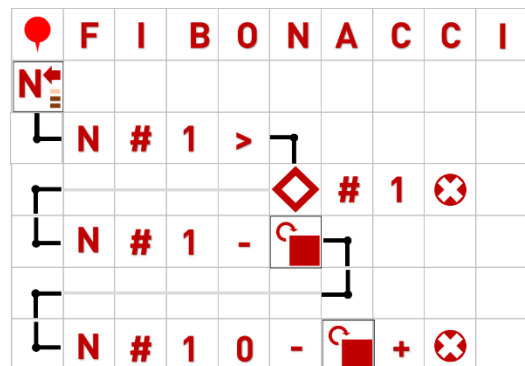
Instrucción	Nombre	Descripción
	Llamada (call)	Extrae un número de la pila que es el índice de la subrutina a invocar. Realiza la llamada a la subrutina. Al finalizar la misma, se retorna al punto de la llamada y se continúa en la dirección en la que se estaba.
	Índice de rutina (routine)	Apila el índice de la rutina actual que está ejecutando.
	Llamada recursiva (recCall)	Realiza la llamada a la propia subrutina. Es equivalente a hacer routine y luego call.

El siguiente programa resuelve el valor N-ésimo de Fibonacci.



AUTO-GUÍAS

Una opción por la que se recibirá una bonificación consiste en agregar automáticamente en las casillas vacías una indicación (línea gris) de por dónde cogería la ejecución, además de poner un [return](#) en todas las posibles casillas donde se sepa se llega a un fin de la subrutina. La auto-guía para el ejemplo anterior luciría como se muestra en la imagen.

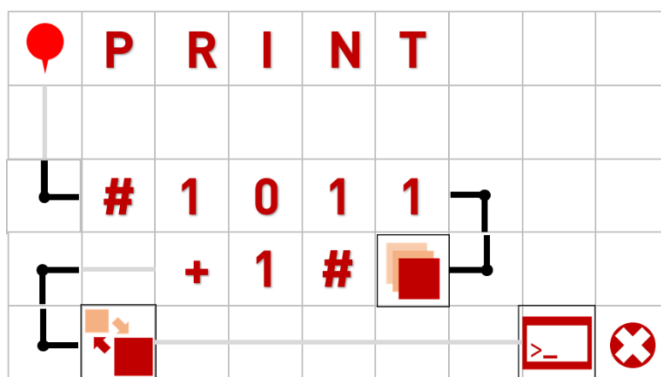


CONSOLA

El simulador tiene una consola que permite imprimir valores. Para imprimir un valor en la consola el programa del robot debe ejecutar la instrucción `print`. Esta instrucción saca un valor de la pila y lo imprime. El texto debe indicar el nombre del robot (algún identificador) y el valor.



El siguiente programa asume que el robot tiene al programa de Fibonacci (expuesto anteriormente) a continuación en la lista de programas. Para imprimir el Fibonacci de 11 bastaría con el siguiente código MATLAN.



En este código primero se deja el valor 11 en la pila (`#1011`), luego se carga el índice de la rutina actual. Éste valor se reemplaza por el siguiente (`#1+`), de esta forma queda en la pila el índice de la rutina de Fibonacci (se había dicho que quedaría a continuación). Luego se realiza la llamada (el `call` saca de la pila el índice de la rutina y deja el valor 11 en el tope como “argumento” de la función Fibonacci). A la salida de la función Fibonacci en la pila queda el valor resultante, por tanto solo queda imprimir.

Note que los registros de la primera fila nunca son alcanzados pero tampoco representan un problema. Las casillas que no son alcanzadas por la ejecución pueden tener cualquier instrucción. Esto puede utilizarse para dejar “comentarios” en el código.



EL ENTORNO SIMULADO

Como se introdujo al principio del documento, los robots inicialmente se ubican en un terreno rectangular dividido en casillas cuadradas. En las casillas se pueden ubicar además objetos con diversas propiedades como tamaño, color, forma, código.

Al inicio se tiene un mapa por defecto vacío y de 10x20 celdas aunque esta configuración puede cambiarse. Una vez creado el mapa se pueden ubicar objetos en él. Cada robot puede cargar un programa (secuencia de rutinas).

Los robots siempre tienen los atributos fijos `shape=bot` y `size=large`.

Para hacer que todos los robots avancen paso a paso (ejecuten su código hasta llegar a una acción física), en el programa del simulador deberá accionarse un botón `advance`. También podría mandar a ejecutar una instrucción a la vez. La ejecución sobre cada robot debe ilustrar gráficamente en cuál instrucción (casilla) se está actualmente. Un botón `play` debería ejecutar automáticamente la simulación con un pequeño intervalo entre cada ronda.

LOS ROBOTS








Cada robot tiene un programa que describe el comportamiento del mismo en todo momento. El lenguaje utilizado será MATLAN. Además, se tienen algunos atributos que expresan el estado de sus sensores.

Como existen varios robots en el mapa, el simulador debe simular la ejecución de las instrucciones de sus programas de forma concurrente. No obstante, en la realidad no ocurrirá así. En cada ronda del simulador todos los robots deben “moverse” un paso. Las instrucciones generales del programa se asumen ejecutan instantáneamente, por tanto lo único que puede significar “toma tiempo” es alguna acción física del robot, por ejemplo, avanzar, girar, soltar, etc. El tiempo de estas acciones será el mismo para todos los robots y aunque tentativamente pudiera ser un segundo, es un posible parámetro de la simulación y por tanto que se pueda ver más lento o más rápido el proceso.

El orden de los robots deberá ser asumido aleatorio en cada “ronda”. De esta forma ningún robot tendrá ventaja sobre otro por el orden en que fueron creados. Una vez establecido el orden en que se simulará el “siguiente paso” de cada robot, se ejecutará en cada uno de ellos el código (desde la última posición en que se había quedado de la ronda anterior) hasta que se ejecute una acción física o se alcance el final del código. Si se llega a una acción física se ejecuta y se pasa al siguiente robot. Si se llega al final del código entonces el robot se queda quieto y comenzará la ejecución desde el principio en la próxima ronda.

Si el robot realiza una acción que no tiene lugar por las características del mapa (por ejemplo, moverse contra un obstáculo) igual “pierde” su turno (como los carritos de control remoto cuando se traban contra una pared pero igual siguen moviendo las ruedas).

En el lenguaje existen distintas constantes enteras para expresar los posibles valores para `color`, `shape`, `direction`.

Instrucciones	Nombre	Valor que apila
	transparent red yellow green blue brown black white	0 1 2 3 4 5 6 7
	nothing sphere box plant robot	0 1 2 3 4
	empty small medium large	0 1 2 3
	north east south west	0 1 2 3
	false true	0 1

Los robots tienen una memoria propia para su programa, variables, rutinas, etc. por lo tanto no existe forma de comunicación entre ellos que no sea a través del entorno.



LA APLICACIÓN

La aplicación visual debe permitir diseñar programáticamente un entorno en el que se ubiquen objetos, robots y se ejecute la simulación.

Todos los programas deberán poder guardarse en ficheros para un futuro uso. El formato en que se salvará los programas será el siguiente:

Un número N con la cantidad de instrucciones

En cada línea siguiente:

Fila Columna Nombre_de_la_Instrucción

Como parte de la especificación del proyecto se le proveerá de una aplicación de ejemplo que permite manejar un conjunto de códigos, y que tiene el proceso de visualización de la matriz con algunos iconos. Usted puede utilizar este proyecto para transformarlo en su propio simulador.

En este documento se exponen un conjunto pequeño de instrucciones, sensores y objetos básicos... No se conforme con ello... Proporcionen nuevos tipos de sensores e instrucciones. Tenga en cuenta que un gran peso de la evaluación lo determina la extensibilidad propuesta. Es decir, cuán fácil es incorporar un nuevo operador, comando, objeto, etc.

Propóngase como meta hacer su solución lo suficientemente extensible como para que se pueda incorporar sin mucha dificultad un posible nuevo operador para la potencia ($2^3 == 8$), o un concepto nuevo.

¡Sea creativo!

No obstante, no puede obviar ninguno de los comandos descritos en este documento puesto que su proyecto será evaluado con códigos propuestos por los profesores.



SOBRE LEGIBILIDAD DEL CÓDIGO

IDENTIFICADORES

Todos los identificadores (nombres de variables, métodos, clases, etc) deben ser establecidos cuidadosamente, con el objetivo de que una persona distinta del programador original pueda comprender fácilmente para qué se emplea cada uno.

Los nombres de las variables deben indicar con la mayor exactitud posible la información que se almacena en ellas. Por ejemplo, si en una variable se almacena “la cantidad de obstáculos que se ha encontrado hasta el momento”, su nombre debería ser `cantidadObstaculosEncontrados` o `cantObstaculos` si el primero le parece demasiado largo, pero nunca `Ob`, `aux`, `temp`, `miVariable`, `juan`, `contador`, `contando` o `paraQueNoChoque`. Note que el último identificador incorrecto es perfectamente legible, pero no indica “qué información se guarda en la variable”, sino quizás “para qué utilizo la información que almaceno ahí”, lo cual tampoco es lo deseado.

- Entre un nombre de variable un poco largo y descriptivo y uno que no pueda ser fácilmente comprensible por cualquiera, es preferible el largo.
- Como regla general, los nombres de variables **no** deben ser palabras o frases que indiquen acciones, como `eliminando`, `saltar` o `parar`.

Existen algunos (muy pocos casos) en que se pueden emplear identificadores no tan descriptivos para las variables. Se trata generalmente de pequeños fragmentos de código muy comunes que “todo el mundo sabe para qué son”. Por ejemplo:

```
int temp = a;  
a = b;  
b = temp;
```

“Todo el mundo” sabe que el código anterior constituye un intercambio o *swap* entre los valores de las variables `a` y `b`, así como que la variable `temp` se emplea para almacenar por un instante uno de los dos valores. En casi cualquier otro contexto, utilizar `temp` como nombre de variable resulta incorrecto, ya que solo indica que se empleará para almacenar “temporalmente” un valor y en definitiva todas las variables se utilizan para eso.

Como segundo ejemplo, si se quiere ejecutar algo diez veces, se puede hacer

```
for (int i = 0; i < 10; i++)  
    ...
```

en lugar de

```
for (int iteracionActual = 0; iteracionActual < 10; iteracionActual++)  
    ...
```

Para los nombres de las propiedades (*properties* en inglés) se aplica el mismo principio que para las variables, o sea, expresar “qué devuelven” o “qué representan”, solo que los identificadores deben comenzar por mayúsculas. No deben ser frases que denoten acciones, abreviaturas incomprensibles, etc.

Los nombres de los métodos deben reflejar “qué hace el método” y generalmente es una buena idea utilizar para ello un verbo en infinitivo o imperativo: Agregar, Eliminar, ConcatenarArrays, ContarPalabras, Arranca, Para, etc.

En el caso de las clases, obviamente, también se espera que sus identificadores dejen claro qué representa la clase: Robot, Obstaculo, Ambiente, Programa.

COMENTARIOS

Los comentarios también son un elemento esencial en la comprensión del código por una persona que lo necesite adaptar o arreglar y que no necesariamente fue quien lo programó o no lo hizo recientemente. Al incluir comentarios en su código, tome en cuenta que no van dirigidos solo a Ud., sino a cualquier programador. Por ejemplo, a lo mejor a Ud. le basta con el siguiente comentario para entender qué hace determinado fragmento de código o para qué se emplea una variable:

```
// lo que se me ocurrió aquel día
```

Evidentemente, a otra persona no le resultarán muy útiles esos comentarios.

Algunas recomendaciones sobre dónde incluir comentarios

- Al declarar una variable, si incluso empleando un buen nombre para ella pueden quedar dudas sobre la información que almacena o la forma en que se utiliza
- Prácticamente en la definición de todos los métodos para indicar qué hacen, las características de los parámetros que reciben y el resultado que devuelven
- En el interior de los métodos que no sean demasiado breves, para indicar qué hace cada parte del método

Es cierto que siempre resulta difícil determinar dentro del código qué es lo obvio y qué es lo que requiere ser comentado, especialmente para Ud. que probablemente no tiene mucha experiencia programando y trabajando con código hecho por otras personas. Es preferible entonces que “por si acaso” comente su código lo más posible.

Otro aspecto a tener en cuenta es que los comentarios son fragmentos de texto en lenguaje natural, en los cuales deberá expresarse lo más claramente posible, cuidando la ortografía, gramática, coherencia, y demás elementos indispensables para escribir correctamente.

Todos estos elementos son importantes para la calidad de todo el código que produzca a lo largo de su carrera, pero además **tendrán un peso considerable en la evaluación de su proyecto de programación.**