Versión de JAVA

Introducción

Por qué programamos Karel

Programar un ordenador en un lenguaje como Pascal, requiere un secuenciamiento preciso de los pasos, uno detrás de otro, escogiendo qué pasos hay que seguir en cada caso, y controlando la repetición de ciertos pasos, en el proceso de resolución de un problema. Aunque esta precisión se requiere para las operaciones sin razonamiento de las computadoras, es extraña a los humanos. Los humanos somos mucho menos rígidos en nuestro comportamiento y podemos retroceder elegantemente si nuestros pasos no parecen llevar a la consecución de un objetivo. Debido a que son diferentes las habilidades de las computadoras y lo humanos, expresar la solución de un problema en instrucciones que una computadora puede seguir está comprobado que es difícil para mucha gente. Para conocer estos conceptos, nosotros empezaremos programando el Robot Karel. Karel es una herramienta de aprendizaje que presenta los conceptos de una forma visual, lo cual es menos abstracto que programar en un lenguaje como Pascal o C. El Robot Karel fue introducido por Richard Pattis en

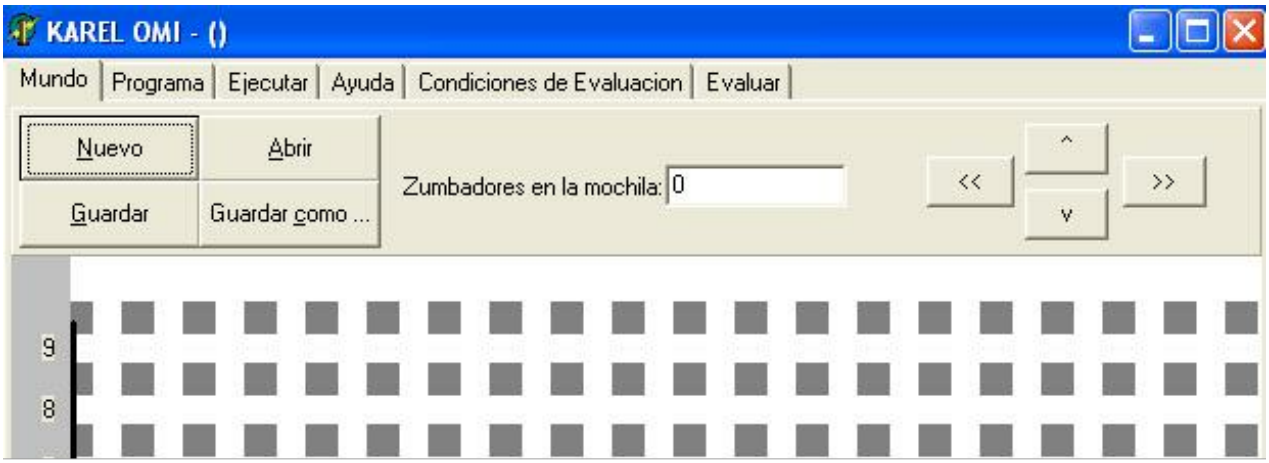
su libro Karel the Robot: A Gentle Introduction to the Art of Programming with Pascal, John Wiley & Sons, Inc., 1981.

Nosotros programaremos Karel, un Robot simple que vive en un mundo simple. Debido a que Karel y su mundo son simulados, ¡nosotros podemos realmente ver los resultados de un programa en acción ! El lenguaje con el que programaremos Karel es una versión especial de Pascal, por lo tanto, la mayor parte de lo que aprendamos, podrá ser aplicado directamente al lenguaje de programación estándar Pascal.

Subir

El mundo de Karel

Karel puede orientarse en una de las cuatro direcciones: Este, Oeste, Norte y Sur. Sólo gira 90º cada vez, por tanto no puede orientarse hacia en NordEste, por ejemplo. En el mundo de Karel, las calles van de Este a Oeste, y son numeradas comenzando por 1. No hay números de calle igual a 0 o negativos. Las avenidas van de Norte a Sur, y también están numeradas empezando por 1. Tampoco hay números de avenida igual a 0 o negativos. Se le llama esquina a la intersección de una calle con una avenida. Karel va de una esquina a la siguiente en un solo movimiento. Ejecuta el programa Karel.exe de la carpeta KarelOMI . Se iniciará el simulador del Robot. Ahora deberías ver la ventana de abajo .

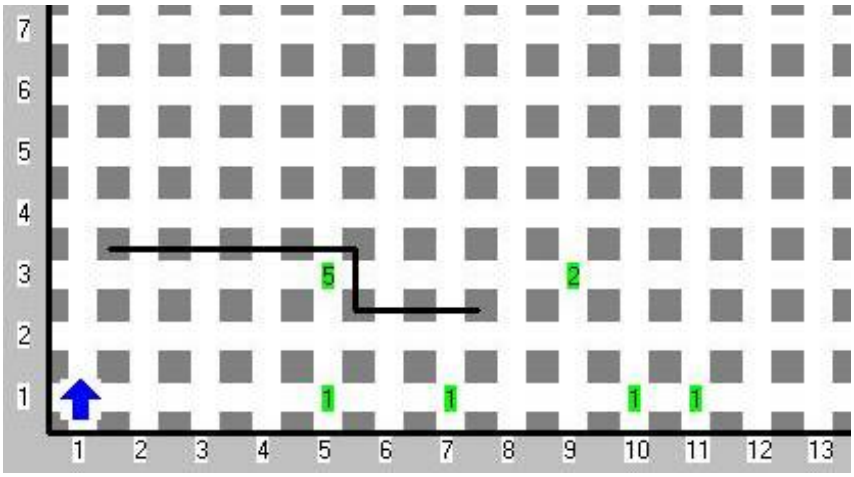


Esta ventana muestra las calles y avenidas que usa Karel para desplazarse. Primero debemos inicializar (o crear) el mundo que Karel va a ocupar. La idea es que puedas introducir algunos elementos en el mundo inicial de Karel.

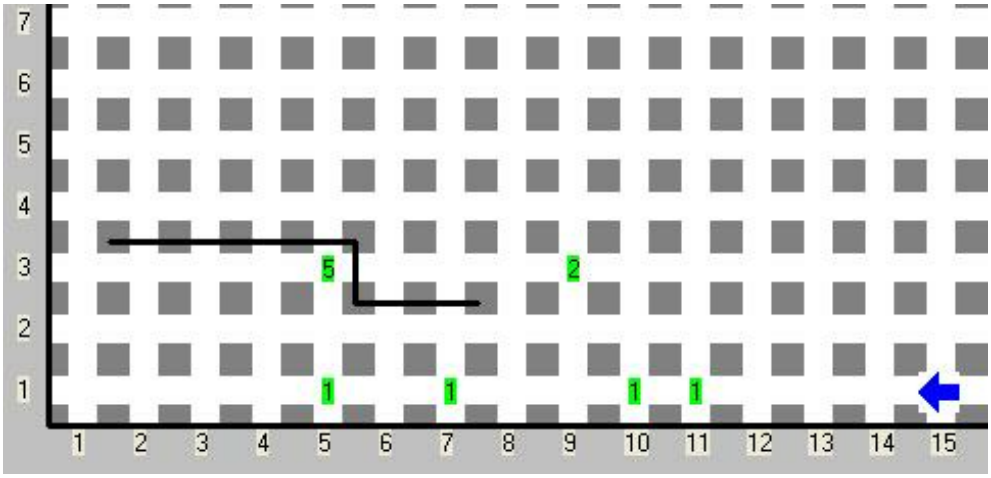
Puedes colocar y quitar muros en el Norte, Sur, Este u Oeste del cursor dando click con el botón izquierdo del ratón en la intersección de las calles correspondientes. Los muros que limitan las calles y avenidas no se pueden quitar, éstos son los que previenen que Karel se salga del mundo. **Prueba a introducir algunos muros para ver que aspecto tienen.**

Otro elemento de interés en el mundo de Karel son los zumbadores. Un zumbador es una forma de marca que Karel puede escuchar sólo cuando se encuentra en la misma esquina que el zumbador. Karel tiene una mochila que puede utilizar para poner los zumbadores que vaya cogiendo. También puede hacer lo contrario, es decir, sacar los zumbadores de su mochila y depositarlos en las esquinas por las que va pasando. Puedes ajustar el número inicial de zumbadores en cada esquina dando click con el botón derecho del ratón en la calle deseada y seleccionando el número de zumbadores deseados (para colocar entre 10 y 99 zumbadores, selecciona la opción N zumbadores y escribe el número deseado).

Prueba a poner algunos zumbadores para ver como se visualizan en el mundo. Crea el mundo inicial que se muestra a continuación. Dado que hemos realizado todo el trabajo necesario para crear un mundo para Karel, ¡vamos a guardarlo ! Pulsa sobre el botón Guardar, ve a tu directorio particular y guarda el mundo como "NuevoMundo.mdo"



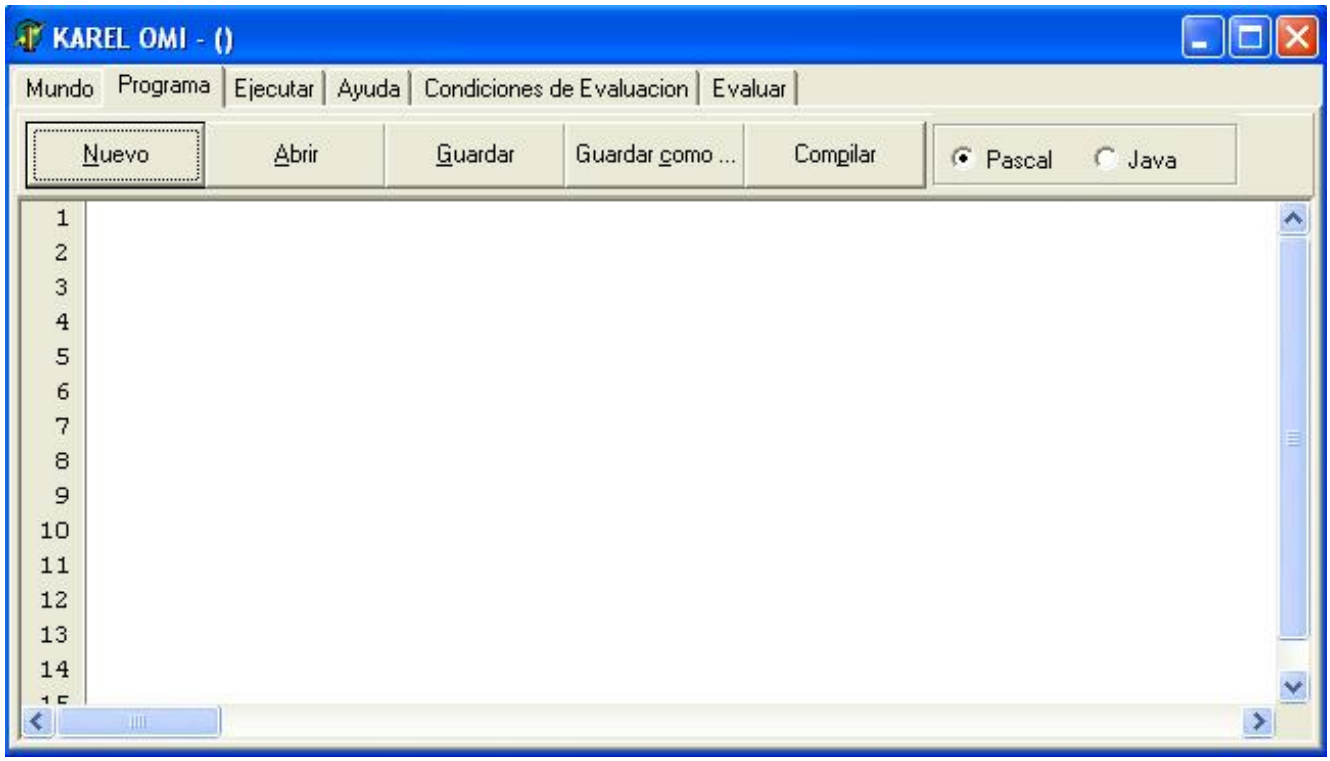
Finalmente, ¡Karel tiene su sitio en el mundo! Mueve el cursor del ratón hacia la esquina de la Avenida 15 y Calle 1 y da click con el botón derecho del mouse, situa el puntero del ratón en la opción "Situar a Karel" y elige "Orientado al Oeste". Ahora deberías visualizar el mundo de abajo. Haz de nuevo click sobre el botón "Guardar" , para almacenar los cambios.



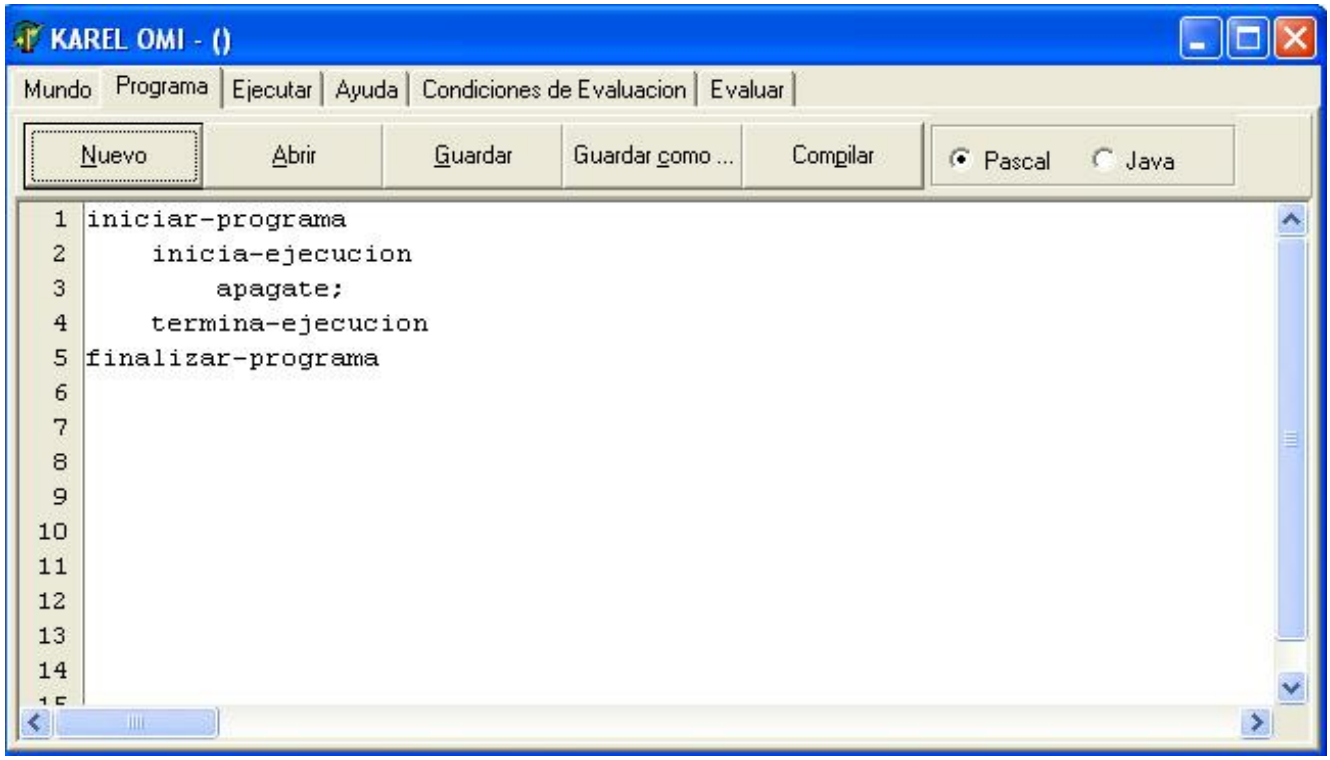
Subir

Programando Karel

Antes de poder empezar, necesitamos dar a Karel un programa (serie de instrucciones) a seguir. Después de todo, ¡es sólo un Robot!. Pulsa sobre la pestaña "Programa" Ahora deberías ver una ventana con el aspecto de esta de abajo. La zona que está vacía es donde escribiremos y veremos el programa de Karel.

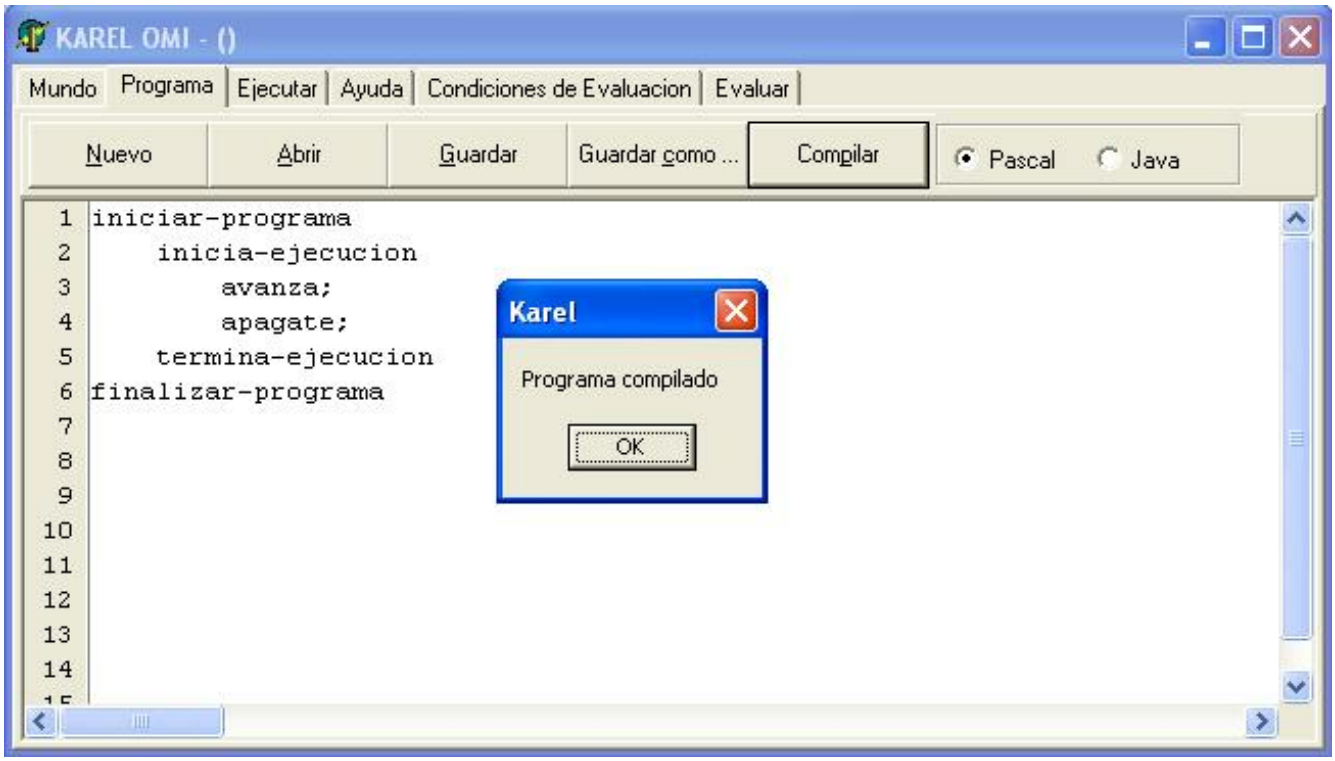


Pulsa en el botón "Nuevo ". Un esqueleto del programa será creado automáticamente. Se creará un programa inicial. Este programa inicial contiene los comandos básicos que son necesarios en cada programa. Ahora deberías ver una ventana como la siguiente.

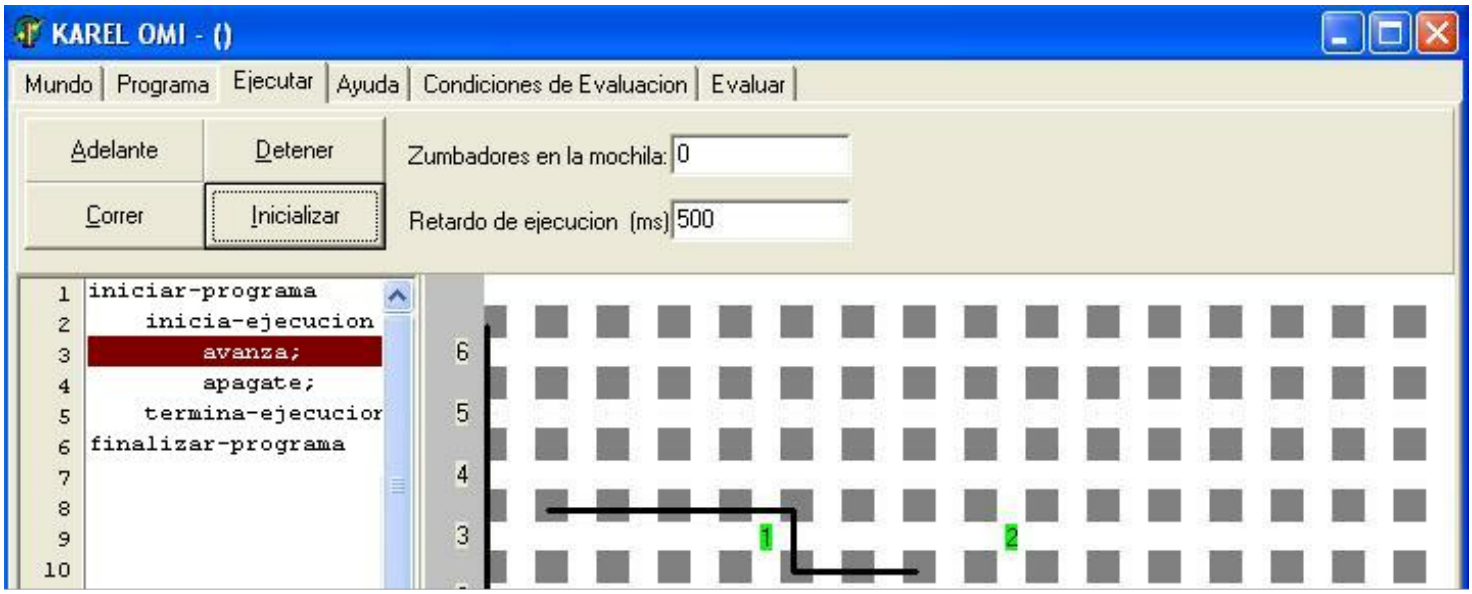


Date cuenta de que el programa anterior sólo le dice a Karel que se apague. Antes de apagarlo, vamos a mandarle algunas tareas. La orden "avanza" le dice a Karel que se mueva hacia adelante una esquina. Escribe "avanza;" antes de"apagate;" . Date cuenta de que el punto y coma se usa para separar órdenes (tal como en

Pascal). Ahora pulsa el botón "Compilar". Si no has cometido ningún error, tu programa tendrás el aspecto del de la ventana siguiente:



Pulsa la pestaña de Ejecutar, y después haz click en el botón Inicializar . (Ejecutar inicia la ejecución (correr) de nuestro programa. Al Inicializar, se muestra el mundo que habíamos creado previamente (NuevoMundo.mdo)). Ahora deberías ver la ventana de abajo (asumiendo que el fichero “NuevoMundo.mdo” que creaste está todavía abierto. Si no, pulsa en la pestaña “Mundo”, y pulsando en el botón “Abrir”, selecciona el fichero “NuevoMundo.mdo” que guardaste en el punto anterior).



Cuando des click en adelante podrás darte cuenta que la instrucción "avanza;" se colorea de rojo. En este punto haz click en “Adelante” para que Karel realice esa primera orden del programa. ¡Wow, Karel se ha movido solo ! Date cuenta que la instrucción en rojo ahora es "apagate;". Haz click sobre “Adelante”de nuevo. Ahora el programa terminará correctamente. Si quieres probarlo otra vez, antes tendrás que pulsar sobre el botón "Inicializar".

Ejercicio 1 : Escribe un programa de Karel para que se mueva a la esquina de la 1ª Calle con la 1ª Avenida y se desconecte, asumiendo que empieza en la esquina de la Calle 15 y la Avenida 15 con orientación hacia el Oeste. Guarda el programa con el nombre “primerPrograma.txt”. Como mundo utiliza el guardado anteriormente con el nombre “NuevoMundo.mdo”.

¡¡ Inténtalo !!

Subir

Comandos básicos de Karel

Hay cinco comandos básicos para Karel, estos son:

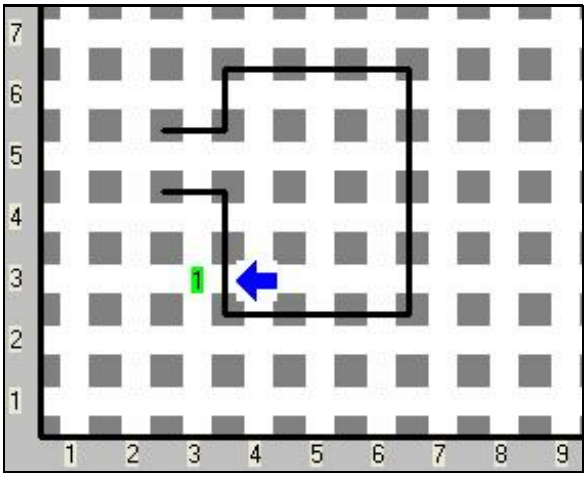
1. **avanza** (avanza una esquina)

2. **gira-izquierda** (gira a la izquierda)
3. **coge-zumbador** (coge un zumbador)
4. **deja-zumbador** (deja un zumpador)
5. **apagate** (desconéctate)

La salud de Karel

Andar entre muros no es bueno para un robot, por lo tanto Karel tiene algunos mecanismos salvavidas dentro de él. Si un programa le dice a Karel que se mueva aunque haya un muro delante de él, él dirá que hay un error y no realizará la acción. Lo mismo ocurrirá si le decimos que coja un zumbador en una esquina y no existe ninguno. Las únicas órdenes que siempre lleva a cabo sin importar la situación en la que se encuentre son gira-izquierda y apagate. Cuando Karel nos dice que hay un error, no tenemos que echarle la culpa, sino que probablemente habremos escrito mal alguna instrucción.

Ejercicio 2 : Cada mañana Karel se levanta de la cama y tiene que recoger el periódico, representado por un zumbador, que está en el porche de la casa. Escribe un programa que ordene a Karel que recoja el periódico y lo lleve de vuelta a la cama. La situación inicial es la de la imagen de abajo, y la situación final debe tener a Karel de vuelta en la cama (misma esquina, misma dirección que cuando empezó) con el periódico (zumbador en su mochila). Crea un mundo como el de la imagen y guárdalo como “periodico.mdo”.



Subir

Sentencias de Control de KAREL

Las sentencias de control se usan para elegir qué hacer, y/o cuantas veces hacerlo. Sin embargo, por si solos no

causan que ocurra algo. Simplemente controlan la ejecución de otras sentencias o fragmentos de código. A continuación se lista una serie de sentencias de control de Karel:

- **Si/Entonces**
- **repite/Nº de veces**
- **Si/Entonces/Sino**
- **Mientras/Hacer**

Siempre a continuación de las sentencias de control **entonces** , **veces**, **sino**, y **hacer** van seguidas de la palabra **inicio** para iniciar el grupo de sentencias a ser realizado. El grupo de sentencias va seguido de la palabra **fin**. Utilizamos el par **inicio/fin** para un sólo parámetro o para varios, no importa el número de ellos. El compilador devolverá una advertencia en caso de que no se siga este estilo.

Nota: Un punto y coma no debe estar a continuación de un **inicio**, pero sí es necesario a continuación de un **fin** (excepto cuando precede una sentencia sino).

Subir

La sentencia Si/Entonces

En el Ejercicio 1 asumimos que Karel estaba orientado hacia el Este. ¿Y si supiéramos que cuando se inicia está orientado hacia el Oeste o hacia el Sur ? A veces necesitaremos girar primero tres veces, y a veces no. En este caso, la sentencia de control **si/entonces** es lo que necesitamos en nuestro programa. Aquí hay un ejemplo de como se debe escribir:

```
...

si orientado-al-sur entonces inicio

  gira-izquierda;

  gira-izquierda;

  gira-izquierda;

fin ;

...
```

Las líneas " ... " significan que pueden haber otras sentencias antes o después de la sentencia **si** . Nos da igual en esta explicación ya que no hay restricciones en cuanto a lo que hay antes o después de la sentencia **si**.

La forma más general de la sentencia **si** es:

...

si xxx entonces inicio

  yyy

fin ;

...

donde xxx es una condición y yyy es cualquier número de sentencias a ejecutar si la condición **si** es verdadera.

Subir

Condiciones que puede detectar Karel

Date cuenta de que la condición orientado-al-sur, en el fragmento de programa de la página del si /entonces

...

si orientado-al-sur entonces inicio

  gira-izquierda;

  gira-izquierda;

  gira-izquierda;

fin ;

...

La condición es una función de la situación actual de Karel, a medida que se ejecuta el programa. Si Karel está actualmente orientado hacia el Sur, el valor de la función orientado-al-sur será verdadero, y el conjunto de sentencias entre el par inicio/fin, se ejecutará. De otra manera, el valor orientado-al-sur será falso y el bloque de sentencias se saltará. Karel comprende cualquier función booleana que comprueba su situación actual. Aquí hay un listado:

frente-libre	**junto-a-zumbador**	**orientado-al-este**
frente-bloqueado	**no-junto-a-zumbador**	**orientado-al-oeste**
izquierda-libre	**algun-zumbador-en-la-mochila**	**no-orientado-al-norte**
izquierda-bloqueada	**ningun-zumbador-en-la-mochila**	**no-orientado-al-sur**
derecha-libre	**orientado-al-norte**	**no-orientado-al-este**
derecha-bloqueada	**orientado-al-sur**	**no-orientado-al-oeste**

"libre" significa que no hay ningún muro, mientras que "bloqueado" significa que hay un muro en esa dirección. Karel puede detectar si hay o no algún zumbador en la esquina en la que se encuentra actualmente, así como detectar si tiene algún zumbador en la mochila o no. También tiene una brújula para detectar hacia qué dirección está orientado.

Por si fuera poco podemos unir dos o más funciones booleanas con los operadores lógicos Y, O, y NO.

Podemos ver la síntaxis de los operadores y sus valores con las siguientes tablas:

Operador "y"

Síntaxis: función-booleana1 y función-booleana2

Valor de la función 1	Valor de la función 2	Resultado final
falso	falso	falso
falso	verdadero	falso
verdadero	falso	falso
verdadero	verdadero	verdadero

Operador "o"

Síntaxis: función-booleana1 o función-booleana2

Valor de la función 1	Valor de la función 2	Resultado final
falso	falso	falso
falso	verdadero	verdadero
verdadero	falso	verdadero
verdadero	verdadero	verdadero

Operador "no"

Síntaxis: no función-booleana

Valor de la función	Resultado final
falso	verdadero
verdadero	falso

Los operadores "y" y "o" se aplican sobre dos funciones y el operador "no" solo sobre una. Lo mejor de los operadores lógicos es que si ponemos dentro de un par de paréntesis las funciones con el operador lógico, entonces toda la operación se vuelve una función booleana, por lo que podemos aplicar más operadores lógicos sobre ella. ¿No quedó claro? Revisa estos ejemplos:

(orientado-al-norte o orientado-al-sur) y frente-libre

(no (frente-bloqueado y junto-a-zumbador)) o orientado-al-este

no (junto-a-zumbador y frente-libre)

Ten mucho cuidado de colocar bien los paréntesis, ya que si no lo haces, no te marcará error, pero seguramente hará algo extraño que tu no quieres que haga.

Por ejemplo, si queremos que Karel avance si está viendo al norte y el frente esté libre, podemos hacer lo siguiente:

```
...

si orientado-al-norte y frente-libre entonces inicio

  avanza;

fin;

...
```

Ejercicio 3 : Escribe un programa de Karel que haga que Karel esté orientado al Norte, desde cualquier dirección inicial, y a continuación se apague. Debería terminar en la misma intersección en la que empezó.

¿Alguna sugerencia?

En un mundo nuevo inserta el ejemplo visto arriba para el caso en que Karel está orientado hacia el Sur. Pero, ¿qué ocurre cuando no está orientado hacia el Sur?. ¡Podría estar también orientado hacia el Norte o el Este !. Entonces necesitas dos sentencias **si** adicionales, en las cuales se especifique que es lo que debe hacer Karel en esas situaciones.

Modifica el mundo inicial de Karel para probar cualquiera de las 4 direcciones de inicio, y para cada una de ellas vuelve a ejecutar el programa.

En esta sección hemos visto como usar las sentencias de control para adaptar Karel a cada situación.

Subir

La sentencia repetir/veces

En el ejercicio 2, tenías que contar la secuencia correcta de pasos para que Karel pudiese resolver el problema. En este caso probablemente no hay una forma más corta de resolverlo. Sin embargo, en algunos problemas hay aspectos del problema que tienen una naturaleza repetitiva. Por ejemplo, para ir de la esquina 15 a la 1 tendríamos que poner un total de 14 "avanza". ¿Es difícil dar justo con el número correcto?. El lenguaje de programación de Karel ofrece un método mejor, la sentencia de control **repetir**. Se escribe como sigue:

...

repetir xxx veces inicio

  yyy

fin ;

...

donde xxx debe ser un número entero positivo, y yyy representa cualquier número de sentencias de Karel. El problema de los 14 avances podría haberse escrito :

...

repetir 14 veces inicio

  avanza ;

fin ;

...

Ejercicio 4: Asume que Karel está en la esquina de la 8ª Avenida y la 8ª Calle, con el escenario que se ve en la siguiente imagen. Escribe un programa que haga a Karel recoger todos los zumbadores y acabe en la 1ª esquina orientado al Sur.

¿Crees que es muy difícil?

Bien… puede ser que un poco de ayuda venga bien. Si puedes imaginar como hacerlo una vez (el primer zumbador de la esquina 7,7), la sentencia repetir lo hará tantas veces como tú quieras.

Crea un mundo como el de arriba y guárdalo con el nombre "diagonal.mdo". Escribe el programa y guárdalo con el nombre "diagonal.txt". Asegurate de utilizar la sentencia repetir/veces. Prueba el programa en este mundo

inicial. Karel debería terminar en la esquina de la primera Avenida con la primera Calle, y llevando 7 zumbadores en la mochila.

Subir

La sentencia Si/Entonces/Sino

Aquí puedes ver como se escribe una sentencia Si/Entonces/ Sino:

...

si xxx entonces inicio

  yyy

fin

sino inicio

  zzz

fin ;

...

donde xxx es una condición, yyy son sentencias a realizar si xxx es verdadero, y zzz son las sentencias a ejecutar si xxx es falso.

Ejercicio 5 : Karel tiene la tarea de alinear una colección de zumbadores en la primera Calle que ha sido distribuida desigualmente. Empieza en la 1ª Calle y la 15ª Avenida y está orientado al Oeste. Se supone que en cada esquina hay exactamente un zumbador. Sin embargo, Karel puede encontrar 0, 1 o 2 zumbadores en cualquier intersección. Su tarea es asegurar que exactamente hay un zumbador antes de continuar hacia la siguiente esquina. Cuando llega a la esquina 1,1 debe apagarse.

Diseña el mundo inicial para que Karel comience en la esquina 1,15 orientado hacia el Oeste. Karel debe empezar con 15 zumbadores en la mochila. Aleatoriamente sitúa 1 o 2 zumbadores en esquinas a lo largo de la 1ª Calle, y deja alguna sin zumbadores. Guarda tu mundo inicial con el nombre “lineaDeZumbadores.mdo”.

Escribe el programa para hacer que Karel complete la tarea descrita anteriormente. Guarda tu programa con el nombre “lineaDeZumbadores.txt”. Truco: utiliza **repetir/veces**, un **Si/Entonces/Sino**, y un **Si/Entonces**.

Recuerda, ¡no sabemos qué esquinas tienen el número equivocado de zumbadores! Karel debe hacer la tarea

correctamente independientemente del nº de zumbadores que haya en cada esquina.

Subir

La sentencia mientras /hacer

Desde el principio, siempre se nos ha dicho la esquina exacta de inicio. Nosotros queremos programar Karel para que se adapte mejor a su mundo. La sentencia mientras/hacer nos permite repetir pasos mientras se cumple una condición, y esto nos va a permitir programar a Karel para que ¡no sea un chico tan rígido!. La sentencia **mientras/hacer** tiene la forma siguiente:

...

mientras xxx hacer inicio

  yyy

fin ;

...

donde xxx debe ser una condición (una de las funciones booleanas listadas anteriormente), y yyy representa cualquier número de sentencias de Karel. El ejercicio 1 podría haberse escrito de la siguiente manera:

...

mientras frente-libre hacer inicio

  avanza ;

fin ;

...

Esto soluciona el problema de caminar de nuevo a la 1ª Calle, sin importar como de lejos se encuentre de esta Calle.

Ejercicio 6: La tarea de Karel es dejar zumbadores a lo largo de una pista de carreras. Un ejemplo de dicha pista es la de la siguiente imagen. Karel debe dar una vuelta completa y depositar un zumbador en cada esquina a lo largo del camino. Guarda el programa con el nombre “pistaCarreras.txt”. Tu solución debe usar sentencias **mientras/hacer**. Construye el mundo inicial de la siguiente imagen con el nombre “pistaCarreras.mdo”. Asegurate de poner dentro de la mochila suficientes zumbadores para todas las esquinas. El ejemplo requiere 22

zumbadores. Karel debe empezar en cualquier intersección de la pista.

Asegúrate de que tu programa funciona en el mundo anterior, y después prueba tu programa modificando el mundo inicial. También, intenta iniciar a Karel desde diferentes intersecciones a lo largo del camino. ¿Realiza Karel su tarea correctamente **en todos los casos**?

Subir

La sentencia define-nueva-instruccion/como

Hasta ahora, le hemos dicho a Karel exactamente lo que tenía que hacer tal como necesitábamos que lo hiciese. Esto funciona bien, pero te pudiste haber dado cuenta de que siempre se utilizan secuencias de sentencias similares. Un ejemplo es cuando Karel tiene que girar a la derecha, y nosotros le decimos " gira-izquierda; gira-izquierda; gira-izquierda; ". ¿No sería más fácil si le pudiéramos decir simplemente "gira-derecha" ?.

En otras palabras, diciendo " gira-derecha; " Karel giraría tres veces hacia la izquierda para alcanzar nuestro objetivo. Es posible. ¿Estás suficientemente motivado para aprender una nueva sentencia?

Una de las razones de crear nuevas instrucciones, es por evitar escribir tanto. Otra es para documentar mejor cual es nuestro objetivo, cuando nosotros mismos u otra persona lee el programa. Como seguramente te estarás dando cuenta, programar es una tarea extremadamente compleja, y ¡necesitamos toda la ayuda necesaria para hacer las cosas correctamente !

Las sentencias **define-nueva-instrucion** están situadas en un sitio especial dentro de un programa de Karel, justo después de la sentencia iniciar-programa. El siguiente es un programa válido para Karel:

Puedes definir cualquier número de instrucciones nuevas, y después usarlas en el programa donde las necesites. Las instrucciones nuevas pueden contener sentencias de control, si es necesario. Date cuenta de que la nueva instrucción puede también usar una instrucción definida previamente. El ejercicio 5 podría haberse escrito:

Date cuenta de como la instrucción repetir acaba moviendo a Karel una esquina hacia adelante, sin tener en cuenta en nº de zumbadores. Si haces esto 14 veces, ¡estarás en casa !

Ejercicio 7: Re-escribe el programa para el ejercicio 4, pero esta vez puede no haber un zumbador en cada esquina. Guarda tu programa con el nombre "diagonal2.txt". La nueva instrucción debería coger un zumbador en la posición actual, si es que lo hay. Deberías usar esta instrucción para coger todos los zumbadores mientras Karel va a su casa en diagonal. Asegúrate de que tienes el mundo "diagonal.mdo" cargado y probar vuestro programa. Karel debería finalizar el la esquina de la 1ª Calle con la 1ª Avenida, con todos los zumbadores que ha ido cogiendo por el camino, y apagarse.

Ejercicio 8: Escribe un programa que ayude a Karel a escapar de un laberinto que no contiene islas (cuadrados aislados). La salida del laberinto está marcada ubicando un zumbador en la primera esquina que está fuera del laberinto, al lado del muro de la derecha. Una forma de resolver este problema es hacer que Karel avance a lo largo del laberinto siguiendo el muro de su derecha (imagina que está tocando el muro y que nunca puede despegar su mano de él). En la siguiente imagen hay un ejemplo de un laberinto del cual debería ser capaz de salir (no olvides que tu programa debería funcionar en todos los laberintos, no solo en el de la imagen). Guarda tu programa con el nombre "laberinto.txt". Esto parece muy, muy complicado. ¿Puedes darnos un mundo de ejemplo?

Aquí tenéis un mundo inicial de ejemplo. La línea roja muestra el camino que debería seguir Karel para este mundo. Recuerda que no sabes de antemano donde estarán los muros.

Podría hacerse más fácil si definieras unas pocas nuevas instrucciones que hicieran parte del trabajo. Aquí tienes un ejemplo:

...

define-nueva-instruccion sigue-muro-derecha como inicio

{pon tu código aquí}

fin ;

...

Esta instrucción hace que Karel avance correctamente hacia el siguiente segmento de muro. Los diagramas de abajo muestran las 4 situaciones, Karel podría estar en cualquier punto del laberinto. Si **sigue-muro-derecha** resuelve correctamente los 4 casos, entonces has solucionado la parte principal del problema. También deberías definir **gira-derecha**.

Para probar tu programa crea un mundo como el del ejemplo y guárdalo con el nombre “laberinto.mdo”. Una vez te funcione el programa para este mundo, prueba a modificarlo anñadiéndo o quitando muros. ¿Realiza Karel la tarea bien **en todos los casos**?

Situaciones iniciales	Movimientos respectivos

Ahora, antes de terminar con este módulo, algo que es importante decir es que si en algún punto necesitas salir de la instrucción que has definido sin necesidad de terminar con el código de dicha instrucción, siempre puedes usar la instrucción **sal-de-instruccion**

Por ejemplo, en el siguiente código:

...

define-nueva-instruccion no-hagas-nada como inicio

si junto-a-zumbador entonces inicio

sal-de-instruccion;

```
  fin;

  avanza;

fin ;
```

...

Karel no avanzará si está junto a un zumbador.

Esto igual podría haberse hecho con un sino, pero en ocasiones, esta instrucción puede llegar a sernos útil

Subir

Parámetros en instrucciones

Hemos visto ya la sentencia **repetir/veces** que nos ayuda a iterar un bloque de código un determinado número de veces, pero siempre teníamos que colocar un número fijo en la sentencia, ¿te has puesto a pensar que pasaría si por ejemplo necesitara una instrucción que volteara a Karel 180°? Pues la respuesta natural sería "haz una instrucción que haga que Karel gire dos veces". Pero... ¿crees que sería posible usar la instrucción que hicimos anteriormente **gira-derecha**? Si existiese alguna forma de que en vez de poner 3 en la sentencia **repetir/veces** pusiesemos un número variable, podríamos usar la instrucción tanto para girar a la derecha como para dar media vuelta.

¡Pues sí existe! Primero retomemos el código para girar a la derecha:

...

```
define-nueva-instruccion gira-derecha como inicio

  repetir 3 veces inicio

    gira-izquieda;

  fin;

fin;
```

...

Ahora, todas las nuevas instrucciones declaradas pueden además llevar un parámetro, ¿pero que es un parámetro?, pues es un numerito que le podemos mandar a la instrucción cuando la llamamos, y como cuando

declaramos la instrucción no sabemos con que número la vamos a llamar, reemplazamos el número por una palabra. ¿Alguna vez has oído la frase "los primeros n números"?, pues precisamente eso son los parámetros. Podemos en vez de n poner 1, 2 ó 3, quedando "los primeros 3 números" por ejemplo. Este parámetro puede tener el nombre que sea, siempre y cuando la primer letra no sea un número y el nombre del parámetro no sea el mismo que una palabra del lenguaje, por ejemplo no se puede llamar si, repetir, avanza, etc.

Este parámetro se puede usar en cualquier lugar dentro de la definición de la instrucción, en cualquier sentencia o instrucción que necesite un número (justo como la sentencia repetir/veces). Redefinamos ahora la instrucción gira-derecha como la instruccion gira:

```
...

define-nueva-instruccion gira (n) como inicio

  repetir n veces inicio

    gira-izquieda;

  fin;

fin;

...
```

De esta forma si escribimos en nuestro código "gira(3);" Karel girará a la derecha, si escribimos "gira(2);" dará media vuelta, si escribimos "gira(1);" girará a la izquierda y si escribimos "gira(0);" no hará nada.

Aqui puedes ver como se escribe una instrucción con un parámetro en general:

```
...

define-nueva-instruccion xxx (yyy) como inicio

  zzz

fin;

...
```

donde xxx es el nombre de la instrucción, yyy es el nombre del parámetro y zzz es cualquier número de instrucciones.

Ejercicio 9: Escribe una nueva instrucción que avance a Karel el número de veces que se le mande como parámetro. Debes de evitar que Karel choque con alguna pared.

¿Tienes dudas? Usa como base el código de la instrucción gira.

Subir

Ahora ya debes de estar listo para empezar a resolver los problemas de la sección Introducción

Origen del curso: Curso de Karel OMI