# Tutorial de Karel el Robot
# Olimpiada Mexicana de Informática

Abril 2015. César Cepeda, cesar@auronix.com

Karel es un pequeño robot en forma de flecha que sólo sabe hacer 5 acciones, pero esas 5 acciones le bastan para hacer muchas cosas. La más interesante de ellas es iniciarte en el mundo de la programación. Si aprendes a manejar a Karel, aprenderás lo básico para crear toda clase de aplicaciones, páginas web, juegos de video, aplicaciones para teléfonos, etc.

En este tutorial aprenderás las instrucciones básicas de Karel y la estructura de un programa. La programación es mágica en el sentido que te permite transformar tus ideas en servicios tangibles en el mundo real. Espero que disfrutes este viaje de iniciación y continúes este camino mucho más adelante.

## REQUERIMIENTOS.

- **Karel el robot:** Para seguir los ejercicios mencionados en este tutorial es necesario que tengas instalado el programa de Karel el robot que se utiliza en la Olimpiada Mexicana de Informática. Si aún no lo tienes instalado puedes obtenerlo en http://www.olimpiadadeinformatica.org.mx/OMI/OMI/Material/Karel_el_Robot.aspx
- **Omegaup:** Si deseas probar tus programas y practicar con más ejercicios deberás crear una cuenta en el portal de OmegaUp. Este portal califica tus programas y tienes cientos de ejercicios que puedes resolver. Si aún no tienes cuenta ingresa a https://omegaup.com/ o crea tu cuenta automáticamente al inscribirte en http://www.olimpiadadeinformatica.org.mx

## INTRODUCCIÓN: ¿QUÉ ES UN PROGRAMA?

Lo más importante a entender de las computadoras es que a diferencia de nosotros, ellas son incapaces de tomar decisiones. En cualquier momento o situación, las computadoras siguen una secuencia de instrucciones. A esta secuencia de instrucciones se le llama *programa*.

Cuando tu mamá te dice "*ve a lavarte los dientes*" no es necesario que te explique dónde está el baño, cuál es tu cepillo, el proceso para ponerle pasta o cómo cepillarte los dientes. Esto se debe a que tú posees inteligencia. Sin embargo a un programa o robot sería necesario darle instrucciones precisas sobre, cómo llegar al baño, cómo identificar un cepillo, cuál cepillo tomar, cómo abrir el tubo de la pasta, cómo ponerle pasta al cepillo, etc. Paso por paso habría que haberle explicado cómo realizar el proceso completo.

Un programa entonces es **una secuencia no ambigua de instrucciones comprensibles por la computadora para realizar un proceso**.

Es muy importante notar que las instrucciones deben ser comprensibles por la computadora. Al igual que tú, la computadora no entiende cualquier lenguaje. Si alguien te da instrucciones en español seguramente podrás seguirlas, sin embargo si te dan las mismas instrucciones, de manera precisa en una lengua extranjera, lo más probable es que no sepas hacer nada con ellas. Con las

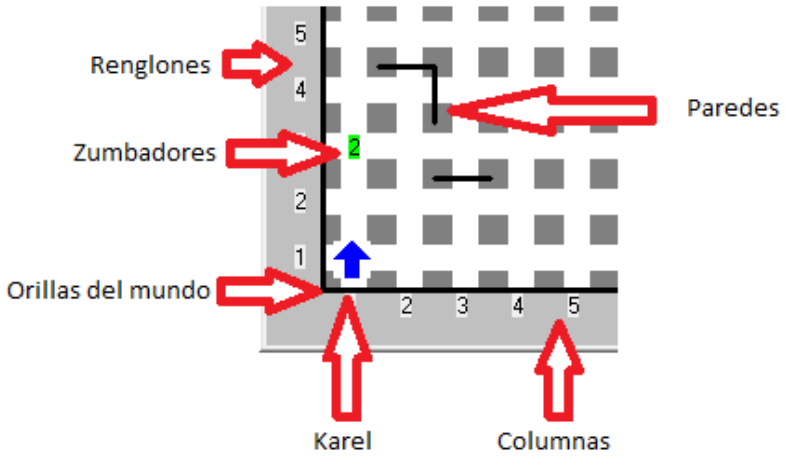
computadoras pasa lo mismo. Para que la computadora pueda seguir tus instrucciones es necesario que las escribas en un programa y un lenguaje que ella entienda.

Karel el robot entiende programas escritos en dos lenguajes distintos Pascal o Java. Ambos lenguajes tienen la misma funcionalidad y el usar uno u otro es cuestión de gusto personal. Para este tutorial utilizaremos Pascal porque las instrucciones se escriben con palabras del español y eso hace que los programas resulten más sencillos de leer.

**EL MUNDO DE KAREL.**

Karel vive en un mundo dentro de la computadora, este mundo es una cuadrícula de 100 renglones con 100 columnas cada uno. Además dentro del mundo de Karel puede haber paredes y *zumbadores*. Los zumbadores son pequeños dispositivos que pueden estar en diversos lugares en el mundo. En cada cuadro de la cuadrícula puede haber desde 0 hasta miles de zumbadores. Los zumbadores se representan como un número que indica la cantidad de dispositivos en cada cuadro.

La siguiente figura muestra un ejemplo del mundo de Karel con algunos de estos elementos:




Ejecuta el programa Karel el robot y ve a la primera pestaña con el título **Mundo** y presiona el botón **Nuevo.** Juega con el mundo, puedes crear paredes haciendo clic con el botón izquierdo del ratón. Para mover a Karel o poner zumbadores haz clic con el botón derecho del ratón sobre el mundo y selecciona la opción que desees.

Construye un laberinto y pon a Karel en algún lugar del mismo. Dibuja una pirámide y coloca a Karel en la cima.

Puedes guardar los mundos que construyas con el botón **Guardar.** Igualmente puedes abrir mundos que hayas creado previamente con el botón **Abrir.**

Una vez que te sientas cómodo manipulando el mundo de Karel puedes pasar a la siguiente sección.

**INSTRUCCIONES DE KAREL.**

Como mencioné en el primer párrafo Karel sólo sabe seguir 5 instrucciones, estas son:

- **avanza:** Al ejecutar esta instrucción Karel avanza un cuadro en la dirección hacia la que está mirando. Pero **cuidado**, si le pides a Karel que avance hacia una pared, tratará de atravesarla y se estrellará ocasionando que tu programa se detenga con un error. ☹
- **gira-izquierda:** Al ejecutar esta instrucción Karel girará un cuarto de vuelta hacia la izquierda. Si ejecutas 4 veces esta instrucción Karel dará una vuelta completa y quedará con la misma orientación.
- **apagate:** Esta instrucción le dice a Karel que debe apagarse y terminar la ejecución. Una vez que Karel llega a esta instrucción se apaga de manera inmediata y el programa se termina.
- **coge-zumbador:** Al ejecutar esta instrucción Karel recogerá un zumbador del suelo y lo guardará en su mochila. Pero **cuidado**, si le pides a Karel que recoja un zumbador y no hay ningún zumbador que recoger, Karel no podrá hacerlo y tu programa se detendrá con un error. ☹
- **deja-zumbador:** Al ejecutar esta instrucción Karel dejará uno de los zumbadores que tiene guardados en su mochila en el suelo. Pero **cuidado**, si le pides a Karel que deje un zumbador y no tiene zumbadores en su mochila para dejar, Karel no podrá hacerlo y tu programa se detendrá con un error. ☹

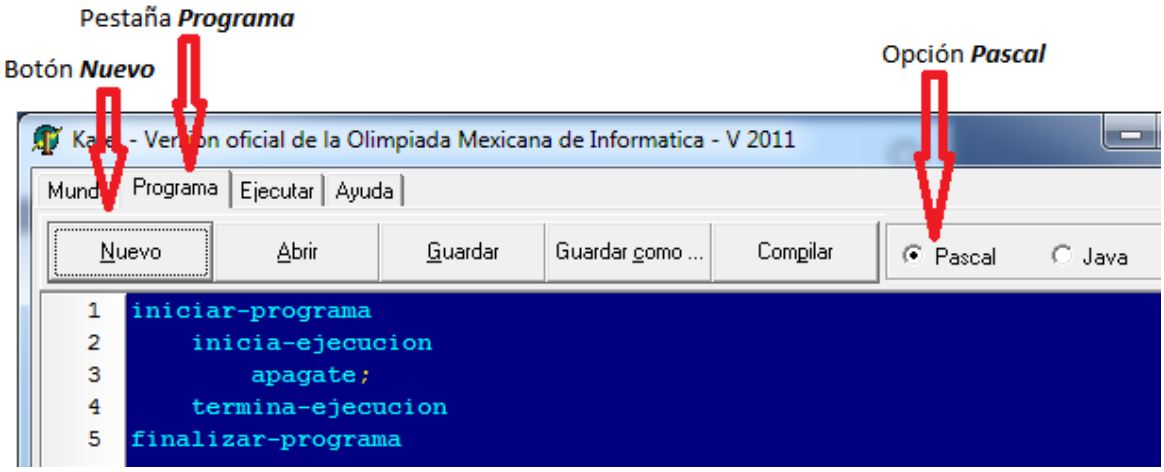
Estas son las 5 instrucciones que Karel entiende y sabe realizar. Sé que parecen pocas, pero te asombrarás de la cantidad de situaciones que Karel puede resolver con esto, te aseguro que son más de las que en este momento puedas imaginar. Sigue y lo verás.

**TU PRIMER PROGRAMA DE KAREL**

Es momento de hacer tu primer programa de Karel. Realiza las siguientes instrucciones:

- Ejecuta la aplicación de Karel el robot (**Karel.exe**).
- Posiciónate en la pestaña de la aplicación con el texto ***Programa***. Asegúrate que esté seleccionada la opción ***Pascal*** en el recuadro a la derecha de los botones y haz clic en el botón que dice ***Nuevo***.

- Tu pantalla debe verse como la que se muestra a continuación.

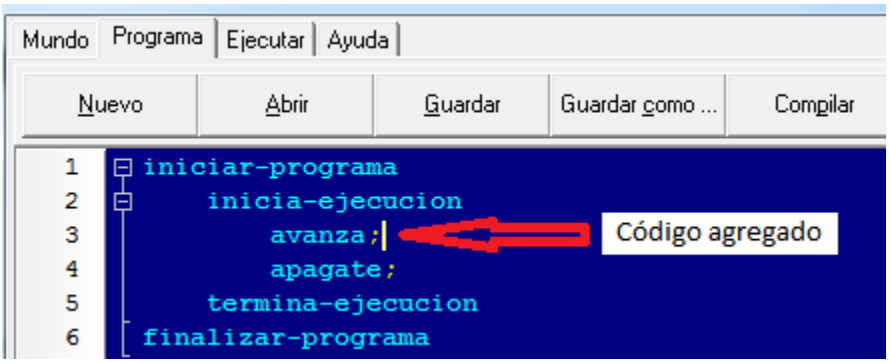



Fíjate que la aplicación te generó automáticamente 5 líneas de código. Estás 5 líneas de código son las mínimas necesarias para que un programa funcione. Estás líneas significan lo siguiente:

- **iniciar-programa** y **terminar-programa:** Estas 2 líneas le indican a Karel dónde inicia y dónde termina el programa. Es inválido escribir instrucciones fuera de este espacio. Estas líneas son **obligatorias**. Por suerte, no es necesario que las recuerdes, ya que cada que presiones el botón ***Nuevo*** la aplicación las escribirá por ti.
- **inicia-ejecucion** y **termina-ejecucion:** Más adelante verás que un programa puede tener muchas instrucciones y no siempre Karel comienza desde la primera o usa todas. Estas 2 líneas le dicen a Karel dónde debe de empezar y dónde debe terminar. Cuando hagas un programa, Karel iniciará ejecutando la línea inmediatamente debajo de **inicia-ejecucion** y se detendrá al llegar a **termina-ejecucion.** Estas dos líneas son **obligatorias.**
- **apagate:** Esta instrucción la mencionamos arriba. Le indica a Karel que al llegar a ella debe apagarse y no hacer nada más. **Fíjate que después de la instrucción aparece un símbolo punto y coma (;).** En el lenguaje Pascal de Karel un bloque de código termina siempre con un (;). Seguramente al inicio es algo que vas a olvidar bastante, pero no te preocupes, después de un tiempo te acostumbrarás y lo harás sin pensarlo.

Agreguemos algo entonces para hacer tu primer programa. Hagamos que Karel dé su primer paso. Colócate al final de la línea 2 (a la izquierda del código hay números que indican el número de línea) y agrega una nueva línea con el siguiente código **avanza;** Si está bien escrito se pondrá

en color azul, de otra forma estará en color amarillo. Tu programa debe verse como se muestra en la siguiente figura:




¿Adivinas qué hará Karel al ejecutar este programa?

Para probar si un programa que escribiste hace lo que tú pensabas hay que realizar los siguientes pasos:

- Estando en la pestaña ***Programa*** haz clic en el botón que dice ***Compilar***.
- Al compilar, Karel revisa que entiende todas las instrucciones que escribiste estén en el formato correcto y no exista ninguna duda.
- Si tu programa es correcto obtendrás una ventana con un mensaje que dice "***Programa compilado***".
- Si tu programa tiene alguna instrucción mal escrita, a la que le falta algo o que Karel no entiende obtendrás un mensaje que describe el error. Por ejemplo, prueba borrar el símbolo (;) al final de **avanza** e intenta compilar nuevamente. Obtendrás el siguiente mensaje "***Se esperaba ";" en la línea 4 posición 9***". Si lees con cuidado el error te dice que hace falta un símbolo (;) en la línea 4, posición 9. "*Pero dónde hace falta es al final de la línea 3*" puedes decir. El símbolo (;) no tiene que ir pegado a la instrucción, puede ir en cualquier lugar entre el final de una instrucción y el inicio de la siguiente. Karel detecta el error cuando ve que después de **avanza** comienza otra palabra (en este caso **apagate**) y no encontró ningún símbolo (;).
- Una vez que compilaste correctamente tu programa ve a la pestaña ***Ejecutar***. Esta pestaña muestra del lado izquierdo tu programa y del lado derecho el mundo de Karel (incluido Karel).
- Para ejecutar tu programa haz clic en el botón ***Inicializar*** para preparar la ejecución y posteriormente haz clic en el botón ***Correr***.

- ¿Qué hizo Karel? Quizá ya habías adivinado. Karel inició su ejecución en la línea inmediatamente debajo de **inicia-ejecucion**, la primer instrucción dice **avanza** por lo que Karel avanza un paso al frente hacia dónde esté mirando. La siguiente instrucción es **apagate**. Al ejecutarla Karel se apaga y obtienes un mensaje que dice "***Ejecución finalizada. Terminación normal***". Esto indica que Karel realizó todas las instrucciones sin ejecutar nada inválido.
- ¡FELICIDADES! Acabas de terminar tu primer programa de Karel.

Ejercicios:

- Regresa a la pestaña ***Mundo*** y posiciona a Karel en un lugar distinto del mundo (recuerda que para posicionar a Karel tienes que hacer clic con el botón derecho del ratón sobre el mundo). Ahora regresa a la pestaña ***Ejecutar*** y vuelve a ejecutar tú programa ¿Qué pasó?
- ¿Recuerdas que dijimos que Karel no podía *avanzar* a través de una pared? Vuelve a la pestaña ***Mundo*** y pon una pared frente a Karel o posiciona a Karel junto de una de las paredes que delimitan el mundo viendo hacia ella. Regresa a ***Ejecutar*** y ejecuta tu programa ¿Qué pasó?
- Haz que Karel avance más de una vez, intenta que avance 2, 3, 4 o la cantidad de pasos que tú quieras.

**PROBRANDO TUS SOLUCIONES EN OMEGAUP**

Como mencioné en la introducción de este tutorial **OmegaUp** es una plataforma que te permite evaluar tus soluciones de Karel. En esta sección vas a aprender cómo hacerlo. Para esta sección es necesario que tengas tu cuenta de OmegaUp creada.

Utilizaremos el programa que acabamos de hacer para calificarlo en OmegaUp.

- Accede a la plataforma de OmegaUp en http://www.omegaup.com
- Pon tus datos de acceso para ingresar a tu cuenta.
- En OmegaUp hay muchos problemas que han sido ingresados por diversas personas los cuales puedes utilizar para resolverlos y evaluarte.
- Una vez que hayas ingresado verás en la parte superior una barra de menús. Haz clic en el menú ***Problemas*** y selecciona la opción ***Problemas.***
- En esta página puedes navegar y buscar los problemas que hay en OmegaUp. Colócate en la caja de texto en la esquina superior izquierda y escribe "**Karel y su primer paso**", luego haz clic en el botón ***Buscar***.
- Quedará sólo un problema en la lista, puedes ingresar a él haciendo clic en el título.
- Al hacer clic te llevará a una página que contiene la descripción de un problema. El problema te pide justo hacer un programa que haga avanzar a Karel una casilla hacia la dirección que está viendo.

- Regresa a la aplicación de Karel y en la pestaña ***Programa*** haz clic en el botón ***Guardar*** y guarda tu código en una ubicación de tu computadora.
- En la página del problema de OmegaUp, baja de la descripción hasta encontrar un botón gris que dice **Nuevo envío**.
- Al hacer clic se abrirá una ventana. Haz clic en el combo de opciones que dice ***Lenguaje*** y selecciona la opción ***Karel (Pascal)***.
- Ahora haz clic sobre el botón que dice ***Seleccionar Archivo*** y selecciona el archivo donde guardaste el programa que hiciste.
- Finalmente haz clic sobre el botón que dice ***Enviar***.
- Recorre la pantalla hacía abajo y espera unos instantes. Si todo es correcto verás un línea sobre el botón ***Nuevo envío*** que muestra tú envío y en la columna ***status*** un pequeño cuadro verde con las letras ***AC*** dentro. En la columna ***Porcentaje*** verás el 100%. Esto significa que tu programa resolvió de manera correcta el 100% de las evaluaciones que se hicieron.
- Si no obtienes el cuadro verde quiere decir que algún paso fue mal ejecutado. Te recomiendo que revises las últimas dos secciones del tutorial para encontrar en dónde falló.

**PROBLEMA 2: Karel aprende a regresar a casa**

En este tutorial vamos a resolver 12 problemas, cada problema irá incrementando de dificultad y aprenderás nuevas instrucciones o funcionalidades de Karel. Revísalos tantas veces como quieras hasta que estés seguro que los entendiste. Haz pruebas, cambios, experimenta. Te recomiendo que no avances al siguiente problema hasta que no tengas completamente dominado el problema en el que estás.

Todos los problemas están disponibles en la plataforma de OmegaUp.

- Accede a OmegaUp.
- Ingresa a la página de búsqueda de problemas (recuerda, menú ***Problemas*** opción ***Problemas***).
- Busca el problema con el nombre ***Karel aprende a regresar a casa***
- Lee el problema y trata de entender lo que se te pide. **NOTA: Los problemas de OmegaUp han sido redactados por varias personas, por lo mismo el estilo de redacción varia y es algo a lo que debes acostumbrarte. En este problema por ejemplo llaman calles y avenidas a los renglones y columnas del mundo. Igualmente llaman esquinas a cada cruce de un renglón y una columna. Conforme leas varios estilos te sentirás más cómodo y te resultarán menos confusos.**

¿Entendiste lo que te piden en el problema? Si no es así, te recomiendo que hagas una pausa e intentes primero entender qué es lo que se te pide que hagas. Si tras un segundo intento sigues con dudas, continúa leyendo.

En el problema mencionan que Karel quiere regresar a su casa, también dice que la casa de Karel está 3 posiciones arriba de la posición en la que empieza. Por último en la sección ***Consideraciones*** menciona que Karel siempre inicia con orientación hacia el norte.

Entonces lo que tenemos que hacer es un programa que haga que Karel avance 3 posiciones hacia el norte y se apague.

El problema es muy similar al problema anterior, con la diferencia que en el problema anterior había que dar 1 paso y en este hay que dar tres. ¿Se te ocurre cómo resolverlo? Seguramente sí ☺

Una opción es hacer un programa como el primero pero en vez de poner una instrucción **avanza;** poner 3 instrucciones **avanza;** Otra opción es utilizar el ciclo **repetir.**

**El ciclo repetir:** Las computadoras y los robots son excelentes para realizar una misma operación muchas veces, pueden hacerlo miles de veces sin cansarse, equivocarse o aburrirse. El lenguaje de Karel permite que le indiques a Karel que repita una secuencia de instrucciones muchas veces sin necesidad de estar escribiendo muchas veces lo mismo. Para lograr esto se utiliza el ciclo **repetir**. Su estructura es la siguiente:

```
repetir <cantidad> veces
inicio
    <instrucciones que quieras repetir>
fin;
```

De aquí en adelante cuando escriba un código y escriba algo entre los símbolos menor qué (<) y mayor qué (>), significa que lo que está entre dichos símbolos debe ser sustituido por algún valor o secuencia de instrucciones dependiendo del problema.

Para este problema queremos que Karel avance 3 posiciones entonces podemos utilizar el ciclo **repetir** de la siguiente forma:

```
repetir 3 veces
inicio
    avanza;
fin;
```

Observa también las palabras **inicio** y **fin.** El lenguaje Pascal de Karel nos permite definir *bloques* de instrucciones. Para definir un bloque de instrucciones tenemos que poner la palabra **inicio** seguida de las instrucciones del bloque y al final escribir **fin;** En este caso nuestro bloque de instrucciones consta únicamente de la instrucción **avanza;** pero un bloque de instrucciones puede tener tantas instrucciones como sea necesario.

En el código anterior estamos diciéndole a Karel que **repita** 3 **veces** el bloque de instrucciones que sigue a continuación, es decir, **avanza;** Por lo tanto Karel avanzará 3 posiciones.

El programa completo se muestra a continuación, si tienes alguna duda regresa a la sección **TU PRIMER PROGRAMA DE KAREL** para recordar la estructura del programa y como iniciar uno nuevo en la aplicación de Karel.

```
iniciar-programa
      inicia-ejecucion
            repetir 3 veces inicio
                  avanza;
            fin;
            apagate;
      termina-ejecucion
finalizar-programa
```

Comprueba tu programa en la pestaña ***Ejecutar***. Recuerda que antes de ejecutar es necesario ***Compilar*** tu programa con el botón de la pestaña ***Programa***.

Utiliza la plataforma OmegaUp para comprobar tu programa.

**Ejercicios:**

- Utiliza el ciclo repetir para hacer que Karel avance un número distinto a 3.
- ¿Qué pasa si pones un ciclo **repetir** dentro de otro? Busca una forma de hacer que Karel avance 9 posiciones poniendo dos ciclos **repetir** uno dentro del otro. **PISTA:** Los dos ciclos repetir deben hacer 3 repeticiones cada uno.

**PROBLEMA 3. Karel y su primer cerca.**

- Accede a OmegaUp.
- Ingresa a la página de búsqueda de problemas (recuerda, menú ***Problemas*** opción ***Problemas***).
- Busca el problema con el nombre ***Karel y su primer cerca***
- Lee el problema y trata de entender lo que se te pide.

Antes de continuar haz tu mayor esfuerzo para entender qué es lo que el problema te está pidiendo que hagas. Una vez que sientas que ya lo entendiste continúa.

Para resolver este problema usaremos las instrucciones que hemos aprendido hasta ahora y utilizaremos las 3 instrucciones de Karel que nos faltaban. Recordando, estas instrucciones son:

- **coge-zumbador:** Recoge un zumbador de la posición donde se encuentra Karel y lo guarda en su mochila. Recuerda que Karel generará un error si le pides recoger un zumbador en una posición dónde no hay.

- **deja-zumbador:** Deja en el suelo uno de los zumbadores que tiene en la mochila. Recuerda que Karel generará un error si le pides dejar un zumbador y no tiene ninguno.
- **gira-izquierda:** Karel gira un cuarto de vuelta hacia la izquierda, esto es, contrario al giro de las manecillas del reloj.

El problema, en la sección de **Consideraciones**, nos dice que Karel debe construir una cerca con zumbadores en forma de un cuadrado de 3 por 3. También nos dice que Karel **no tiene** zumbadores en su mochila al inicio, pero que en la posición donde inicia hay 8 zumbadores en el suelo (que son los que necesita para construir la cerca de 3x3). Por último nos dice que la esquina inferior izquierda de la cerca debe quedar en la posición inicial de Karel.

A diferencia de los problemas anteriores, para resolver este problema es necesario pasar por dos etapas.

**Etapa 1 (recoger los zumbadores):** El primer paso es recoger los zumbadores del suelo. De otra forma no vamos a tener zumbadores para construir la cerca.

**Etapa 2 (construir la cerca)**: Una vez que tengamos material suficiente entonces tendremos que construir la cerca.

Para la etapa 1 tenemos que recoger 8 zumbadores del suelo, sabemos que para recoger un zumbador podemos usar la instrucción **coge-zumbador** y también conocemos de nuestro problema anterior el ciclo **repetir**. Tenemos entonces todos los elementos para resolver el problema. ¿Se te ocurre cómo? A continuación el código de cómo puede hacerse.

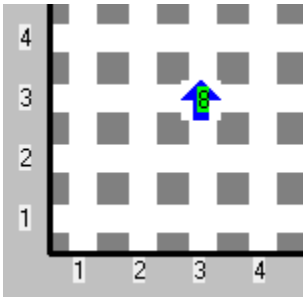
```
repetir 8 veces inicio
   coge-zumbador;
fin;
```

Antes de continuar con la etapa 2, vale la pena comprobar que nuestro programa funciona bien para la etapa 1. Haz un nuevo programa con el código de la caja anterior. El programa completo se muestra a continuación.

```
iniciar-programa
      inicia-ejecucion
            repetir 8 veces inicio
                  coge-zumbador;
            fin;
            apagate;
      termina-ejecucion
finalizar-programa
```

Recuerda que una de las ***Consideraciones*** de este problema es que hay 8 zumbadores en la posición donde inicia Karel. Para poder probarlo necesitas construir un mundo con estas características. Ve a la pestaña ***Mundo*** y sigue las siguientes instrucciones:

- Presiona el botón ***Nuevo*** para borrar cualquier cosa que esté en el mundo y comenzar con un mundo vacío.
- Sitúa a Karel en una posición distinta del renglón1, columna 1. Digamos en la posición 3, 3. Karel debe estar orientado al Norte. Recuerda que para situar a Karel debes hacer clic con el botón derecho del ratón en la posición donde quieres situarlo.
- Ahora debemos poner 8 zumbadores en la posición donde inicia Karel. Para hacerlo de nuevo debes hacer clic con el botón derecho del ratón en la posición donde situaste a Karel y selecciona la opción que dice ***8 zumbadores***.
- Una vez que hayas terminado el mundo debe verse como en la siguiente figura:




Estamos listos para probar el programa. Es buena idea que grabes el mundo que hiciste para no tenerlo que hacer de nuevo en el futuro en caso de que lo necesites.

Ahora aprenderás algo nuevo. Compila tu programa y ve a la pestaña ***Ejecutar***. Presiona el botón ***Inicializar*** pero esta vez en lugar de *correr* el programa lo aprenderás a ejecutar paso a paso. Ejecutar un programa paso a paso te sirve para ver exactamente lo que pasa en tu programa. Conforme aprendas a crear programas más complejos más útil será que puedas revisar paso a paso porciones de tu programa.

El botón ***Adelante*** te permite ejecutar una línea de tu programa. Con el botón ***Adelante*** puedes ir ejecutando tu programa paso a paso. Ejecuta tu programa con el botón ***Adelante*** hasta que termine. Fíjate cómo cambia el número de zumbadores en la posición dónde está Karel y el número de zumbadores en la mochila de Karel cada instrucción que se ejecuta.

Cuando te hayas convencido de que la etapa 1 funciona como se planeó es momento de pasar a la etapa 2. La etapa 2 es construir la cerca. La cerca es un cuadrado, un cuadrado se caracteriza porque sus 4 lados son iguales. Recuerda que la computadora es excelente en repetir muchas veces lo mismo por lo que si queremos hacer un cuadrado, podemos hacer el código para un lado y luego pedirle a Karel que lo repita 4 veces.

Lo anterior nos deja con un nuevo problema. ¿Cómo hacemos un lado? El lado del cuadrado mide 3 casillas, eso quiere decir que si Karel inicia en la primera debe avanzar 2 posiciones para llegar a la última. Entonces para construir un lado de la cerca Karel debe avanzar 2 veces. Además por cada posición que pase Karel debe dejar un zumbador.

Es probable que la explicación anterior no te sea clara la primera vez, si es así te recomiendo que tomes un papel cuadriculado y trates de seguir en el papel la explicación escrita arriba. Si tienes duda no te desesperes, todo irá quedando claro conforme avancemos.

Dijimos entonces que para construir un lado debemos avanzar 2 veces y dejar un zumbador en cada posición a la que avancemos. El código entonces para construir un lado de la cerca se vería como sigue:

```
repetir 2 veces inicio
      avanza;
      deja-zumbador;
fin;
```

Además dijimos que como era un cuadrado bastaba con escribir el código de un lado y repetirlo cuatro veces. Digámosle entonces a Karel que repita las instrucciones para construir un lado 4 veces. El código queda así:

```
repetir 4 veces inicio
      repetir 2 veces inicio
            avanza;
            deja-zumbador;
      fin;
fin;
```

Si juntamos este código con el de la etapa 1 obtenemos el siguiente programa:

```
iniciar-programa
      inicia-ejecucion
            repetir 8 veces inicio
                  coge-zumbador;
            fin;

            repetir 4 veces inicio
                  repetir 2 veces inicio
                        avanza;
                        deja-zumbador;
                  fin;
            fin;

            apagate;
      termina-ejecucion
finalizar-programa
```

¿Qué opinas de este programa? ¿Te parece que resolverá el problema? Compílalo, ve a ***Ejecutar*** y pruébalo. Te recomiendo ejecutarlo paso a paso y analizar lo que sucede.

El programa no funcionó ☹. ¿Se te ocurre cómo arreglarlo? Tu programa recoge los zumbadores para construir la cerca y construye 4 lados, el problema es que en lugar de construirlos en forma de cuadrado, los construye en forma de línea. Al terminar de construir cada lado Karel debe girar a la derecha antes de construir el siguiente lado. Necesitamos entonces hacer que Karel gire al terminar de construir cada lado.

Tenemos otra situación que resolver (los programas suelen ser así, conforme avanzas en su solución se generan nuevas situaciones más pequeñas que hay que resolver), Karel solo sabe girar hacia la izquierda con la instrucción **gira-izquierda**, pero en este caso necesitamos que gira a la derecha. ¿Cómo podemos lograrlo? Una posible solución es girar 3 veces hacia la izquierda, eso será equivalente a girar una vez hacia la derecha.

Agreguemos entonces un giro a la derecha al terminar de construir un lado para arreglar el programa. El código final queda así:

```
iniciar-programa
      inicia-ejecucion
            repetir 8 veces inicio
                  coge-zumbador;
            fin;

            repetir 4 veces inicio
                  repetir 2 veces inicio
                        avanza;
                        deja-zumbador;
                  fin;

                  repetir 3 veces inicio
                        gira-izquierda;
                  fin;
            fin;

            apagate;
      termina-ejecucion
finalizar-programa
```

Vuelve a compilar tu programa y probarlo en la pestaña ***Ejecutar***. Una vez que estés satisfecho con él guarda tu código y utiliza la plataforma de OmegaUp para evaluarlo. Si no recuerdas los pasos, consulta la sección **PROBANDO TUS SOLUCIONES EN OMEGAUP,** al inicio del tutorial.

**Ejercicios:**

- Modifica el programa para que en vez de un cuadrado de 3x3 Karel construya una cerca de 4x4, 5x5 u otros tamaños. **PISTA**: Date cuenta que para construir una cerca más grande Karel necesitará más zumbadores, por lo que tendrás que cambiar el mundo para que haya más zumbadores en la posición donde Karel inicia.

**PROBLEMA 4. K-Básico-Camina hasta la pared**

Al igual que con los ejemplos anteriores, lo primero que hay que hacer es ubicar el problema en la plataforma de OmegaUp para que lo leas.

Ya que lo hagas es muy importante que entiendas que cuando te evalúen no sabrás a cuanta distancia está la pared del punto donde inicio Karel, por lo que el programa que escribas deberá funcionar sin importar que tan cerca o lejos está la pared.

Al igual que en el ejemplo anterior para probar este problema necesitamos construir un mundo. Ve a la pestaña ***Mundo*** y crea un nuevo mundo. Sitúa a Karel en la posición que gustes orientado al norte y construye una pared algunas posiciones arriba. Un posible ejemplo es el que se muestra arriba. Sin embargo recuerda que tu programa debe funcionar sin importar que tan cerca o lejos quede la pared, te recomiendo entonces que construyas la pared a una distancia distinta que la que se muestra en el ejemplo.

Para poder resolver este problema necesitas aprender nuevas herramientas sobre Karel. Además de saber ejecutar 5 instrucciones, Karel sabe hacer preguntas acerca del mundo. Las preguntas que hace Karel todas se responden con un **Sí** o un **No.** Karel sabe hacer muchas preguntas, las iremos aprendiendo por el momento basta con aprender una.

- **`frente-libre:`** `Cuando Karel hace esta pregunta la respuesta será` **`Sí`** `cuando no haya pared frente a Karel, la respuesta será` **`No`** `si hay una pared frente a Karel.`

Pero ¿cómo usa Karel las respuestas a las preguntas que hace? Una forma es utilizar el ciclo **mientras**, la estructura del ciclo **mientras** es la siguiente:

```
mientras <la respuesta a una pregunta sea sí> hacer inicio
    <instrucciones que quieras ejecutar>
```

```
fin;
```

El ciclo **mientras** sirve para repetir una secuencia de instrucciones **mientras** una condición se cumpla, es decir, mientras la respuesta a una pregunta sea **Sí**. ¿Cuál es la diferencia de **mientras** con **repetir**? **Repetir** repite un bloque de instrucciones un número determinado de veces, **mientras** repite un bloque de instrucciones un número no determinado de veces, lo repite **mientras** una condición se cumpla.

Para este problema podemos utilizar el ciclo **mientras** con la pregunta **frente-libre**. El problema nos pide que hagamos que Karel avance hasta encontrar una pared y en ese momento se detenga. Mezclando las dos nuevas instrucciones que aprendiste con **avanza** se obtiene el siguiente código.

```
mientras frente-libre hacer inicio
     avanza;
fin;
```

El código de arriba le indica a Karel que debe **mientras** la respuesta a **frente-libre** sea **Sí** (lo cual significa que Karel no tiene ninguna pared frente a él) debe ejecutar una instrucción **avanza**. El ciclo **mientras** es una de las estructuras más usadas en la programación, por lo que es muy importante que lo entiendas bien. Su funcionamiento es el siguiente:

- Karel pregunta si tiene el **frente-libre**.
- Si la respuesta es **Sí** entonces ejecuta la instrucción **avanza**.
- Al hacerlo vuelve a hacer la pregunta.
- El ciclo se repite una y otra vez hasta que la respuesta a la pregunta **frente-libre** sea **No**, lo cual quiere decir que Karel tiene una pared en frente. En ese momento el ciclo termina y la ejecución continúa con la instrucción inmediatamente debajo del **fin;**

Como puedes observar de la descripción de arriba el código hará que Karel avance hasta encontrarse frente a una pared. El problema te pide que hagas que Karel avance hasta encontrar una pared y se detenga frente a ella. El código completo del programa quedará como sigue:

```
iniciar-programa
     inicia-ejecucion
          mientras frente-libre hacer inicio
               avanza;
          fin;
          apagate;
     termina-ejecucion
finalizar-programa
```

Compila tu programa y haz algunas pruebas. Regresa al mundo y cambia la pared de lugar y observa que pasa. Cuando estés satisfecho con el programa y su funcionamiento accede a OmegaUp y evalúa tu solución.

**Ejercicios:**

- Karel sabe hacer también preguntas sobre su orientación. Karel puede hacer las preguntas **orientado-al-norte, orientado-al-este, orientado-al-sur** y **orientado-al-oeste**. La respuesta a cada una de estas preguntas será **Sí** si Karel está orientado en dicha dirección y **No** en caso contrario. Haz que el programa anterior funcione sin importar la orientación inicial de Karel, haz un ciclo **mientras** que primero se asegure de que Karel está **orientado-al-norte** y después avance hasta la pared. **PISTA:** Necesitarás un ciclo **mientras**, la pregunta **orientado-al-norte** y la instrucción **gira-izquierda.**

**PROBLEMA 5. K-Básico-Camina y recoge**

Accede a OmegaUp y busca el problema para que lo leas.

Este problema te pide que hagas que Karel avance hasta la primera pared igual que el problema anterior. Pero además te solicita que vaya recogiendo los zumbadores que encuentre en el camino, los deje en la posición final y después se apague.

Como en los ejemplos anteriores para resolver este problema es necesario que aprendas algunas cosas nuevas sobre Karel.

Empecemos con 2 nuevas preguntas:

- **junto-a-zumbador:** Karel utiliza esta pregunta para saber si hay algún zumbador en la posición donde se encuentra. La respuesta será **Sí** si hay uno o más zumbadores en la posición donde está Karel, la respuesta será **No** si no hay ningún zumbador.
- **algun-zumbador-en-la-mochila:** Karel utiliza esta pregunta para saber si tiene zumbadores en la mochila. La respuesta será **Sí** si el número de zumbadores en su mochila es mayor a cero. La respuesta será **No** si el número de zumbadores en su mochila es cero.

Además de las dos preguntas anteriores necesitarás una nueva estructura, la estructura **si entonces**. El código de la estructura **si entonces** es el siguiente:

```
si <la respuesta a una pregunta es sí> entonces inicio
      <instrucciones que quieras ejecutar>
fin;
```

La estructura **si entonces** te permite preguntar si una condición se cumple en un cierto momento haciendo una pregunta. Si la condición se cumple entonces Karel ejecutará las instrucciones entre el inicio y el fin. Si la condición **no** se cumple y la respuesta a la pregunta es **No** entonces Karel se

"saltara" todas las instrucciones entre el **inicio** y el **fin** y continuará ejecutando la instrucción inmediatamente debajo del **fin**.

Como en los ejemplos anteriores, para poder probar tu programa es necesario que construyas un mundo con las características que se describen en el problema. El problema tiene un dibujo de un mundo de ejemplo. Ve a la pestaña ***Mundo*** y crea un mundo como el del ejemplo. Recuerda que cuando tu programa se evalúe se utilizarán varios mundos distintos, la pared estará a diferentes distancias y los zumbadores en distintas posiciones.

Recapitulemos lo que el problema nos pide. Tienes que hacer que Karel avance hasta encontrar una pared, eso lo aprendimos a hacer en el ejemplo anterior, en este ejemplo además tenemos que recoger los zumbadores que hay en el camino (recuerda que si Karel intenta recoger un zumbador en una posición donde no hay zumbadores se generará un error) y al final dejar todos los zumbadores en la posición frente a la pared.

Como en el ejemplo anterior este programa se puede dividir en 2 etapas.

**Etapa 1:** Avanzar hasta la pared recogiendo los zumbadores que encuentres.

**Etapa 2**: Dejar todos los zumbadores encontrados en el camino que se guardaron en la mochila.

Para la etapa 1 utilizaremos el código básico del problema anterior y tenemos que agregar algo que nos permita recoger un zumbador cuando encontremos alguno. Lo podemos hacer utilizando la estructura **si entonces** con la pregunta **junto-a-zumbador** y mezclándolo con el código del problema anterior. El código puede quedar como sigue:

```
mientras frente-libre hacer inicio
      si junto-a-zumbador entonces inicio
            coge-zumbador;
      fin;
      avanza;
fin;
```

Te recomiendo analizar con detalle el código anterior. La base es el problema del ejemplo previo que indica a Karel que avance mientras no esté frente a una pared. Además agregamos un código que antes de avanzar pregunta si en la posición que estamos hay algún zumbador. En caso de que lo haya lo recoge, si no hay simplemente avanza. De esa forma Karel sólo intenta recoger zumbadores en aquellas posiciones donde hay un zumbador.

Como en los ejemplos anteriores te recomiendo que antes de continuar pruebes el código de arriba para ver que la primera etapa funciona como se desea. Prueba distintas configuraciones del mundo para asegurarte que siempre funciona.

Para la etapa 2 Karel debe dejar todos los zumbadores que recogió en la posición frente a la pared. Puedes lograr esto utilizando un ciclo **mientras** con la pregunta **algun-zumbador-en-la-mochila.**

Recuerda que cuando Karel recoge un zumbador lo guarda en su mochila, por lo que al llegar a la pared traerá en su mochila todos los zumbadores que recogió.

El ejemplo de código para la segunda etapa queda como sigue:

```
mientras algun-zumbador-en-la-mochila hacer inicio
    deja-zumbador;
fin;
```

Intenta crear un programa juntando el código de las dos etapas para resolver el problema. Pruébalo con diferentes mundos y cuando estés satisfecho utiliza la plataforma de OmegaUp para evaluar tu solución.