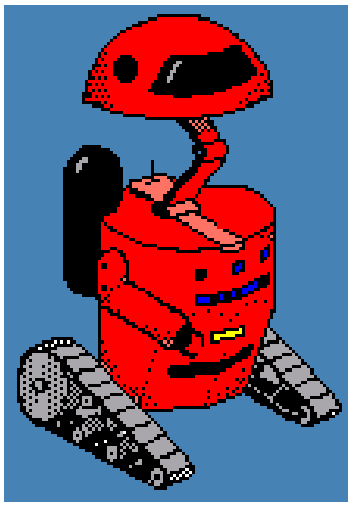
Karel el Robot



Materiales de Apoyo

ESTADO DE VERACRUZ
Problemas de Guanajuato, Jalisco y Aguascalientes
Método D.F. y Estado de Mexico

Febrero 2010.

COMI COMITE MEXICANO DE INFORMÁTICA

Karel el Robot

Índice de contenido

Introducción

En 1986 Honda saca a la luz su primer diseño de lo que en un futuro cambiara al mundo en materia de robótica. Durante estos años Honda a perfeccionado su diseño original y con los grandes adelantos y miniaturización de componentes electrónicos a llegado hoy a lo que en el mundo de la Robótica se conoce como el más gran avance en la búsqueda de crear un Robot cada vez más humanoide. ASIMO es el nombre de su proyecto, este robot ha superado los proyectos desarrollados por muchos otros, entre los que destaca el Instituto de Massachussets o la Universidad de Carnegie Mellon. ASIMO es un pequeño robot de 1.20 m de altura y 43 Kg. de peso que maravillo al mundo cuando Honda lo presento en cadena nacional. ASIMO ah cambiado mucho desde su aparición y lo seguirá haciendo, es un hecho que hoy mismo este lista una versión más ligera, más inteligente, más rápida, más eficiente, económica y sobre todo más preparada para interactuar con el ser humano.

Esta interacción se logra dotando a ASIMO de programas de computadora, que no hacen otra cosa que ejecutar los más complejos algoritmos diseñados por los ingenieros de Honda. Estos algoritmos son puestos en práctica antes de que se agreguen a la memoria de ASIMO, para esto, los ingenieros de Honda necesitan simular las posibles situaciones que ASIMO encontrara en su interacción con el ser humano. Subir una escalera, esquivar un mueble o dar vuelta en un pasillo de una oficina o tomar o dejar un objeto son tareas que para nosotros son sencillas pero para una máquina como ASIMO es sumamente complejo. El ingeniero necesita entonces simular estas situaciones para desarrollar un algoritmo eficiente que las resuelva, este simulador es un programa de computadora que simula un espacio virtual donde ASIMO se desarrollara. Un buen ejemplo de este simulador es un programa desarrollado por el Profesor Richard Pattis, en el Departamento de ciencias de la Computación en la Universidad Carnegie Mellon, su nombre es Karel el Robot.

Karel el robot es un programa que permite escribir algoritmos que resuelven problemas dados en un ambiente gráfico simulado. El lenguaje que se utiliza para poner en practica los algoritmos es el Pascal, que permite una programación estructurada con instrucciones de repetición y de condicionamientos sencillos. Al ambiente gráfico simulado se le llama El Mundo de Karel. Un mundo de Karel esta formado por calles, avenidas y objetos dentro del mundo, estos objetos pueden ser muros por los que Karel no puede pasar; estos muros dan forma a diferentes objetos en el mundo, ya sea, laberintos, escaleras, etc., obstáculos que normalmente encontramos en nuestra vida cotidiana.

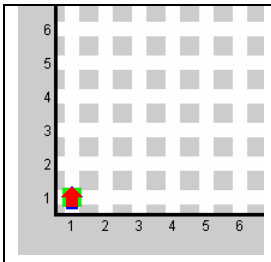
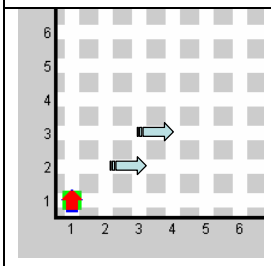
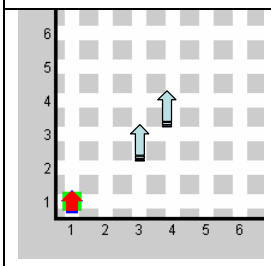
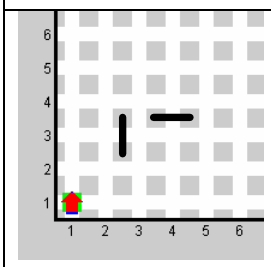
Estos elementos dan forma a simulaciones de situaciones que normalmente un robot como ASIMO puede encontrar en su interacción con el ser humano. Para concluir, ASIMO y su evolución es para muchos investigadores la Biblia de la robótica humanoide, otros consideran que la robótica humanoide no es rentable, el tiempo y la propaganda que Honda recibe gracias a su humanoide lo dirán, mientras tanto el que en otro tiempo fuera fabricante de electrodomésticos y coches es ahora un componente vital de la historia de la robótica. Y los simuladores como Karel el Robot nos permiten desarrollar aquellos complejos algoritmos que estos robots tendrán que ejecutar para interactuar con nosotros.

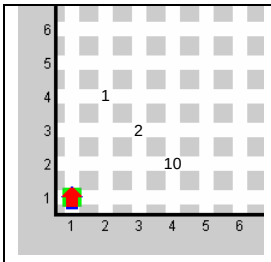
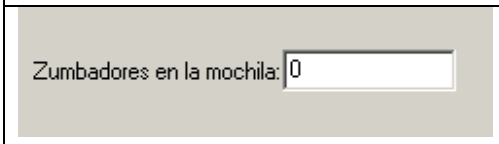
1 Karel el Robot

En esta primera parte pretendemos introducir varios conceptos básicos de programación. Trataremos de brindar la posibilidad de adquirir los principios de un lenguaje de programación estructurado, tales como la *creación de procedimientos*, *condicionales*, *iteraciones*, etc.

1.1 Objetos en El mundo de Karel

Karel el Robot es un programa que simula un mundo virtual de un robot donde el objetivo es la resolución de problemas de carácter logístico. El robot se llama Karel y lo podemos controlar por medio de un algoritmo que primero se diseña y después se captura como instrucciones reconocidas por el programa para llegar a la resolución del problema en cuestión. En el mundo de Karel encontraremos los siguientes elementos:

	Karel. Representado por una flecha en la parte inferior izquierda de su mundo, la dirección de la flecha marca la orientación de Karel.
	Calles. Se extienden desde el muro del Oeste hacia el Oeste indefinidamente.
	Avenidas. Se extiende desde el muro del sur hacia el Norte indefinidamente.
	Muros. Paredes colocadas de manera estratégica para impedir el paso de Karel.

	Zumbadores. Objetos sonoros representados por números. El número 1 representa un zumbador, el número 2 representa 2 zumbadores, el número 10 representa 10 zumbadores.
	Una Mochila. Una bolsa virtual que le sirve a Karel para almacenar los zumbadores que carel tenga que recoger o los que por defecto debe de llevar en la mochila para colocar el en mundo.

1.2 Componentes de Karel

El estado inicial de Karel en el mundo es el origen, que es la esquina inferior izquierda con orientación hacia el norte. Es un robot que cuenta con tres sensores que detectan muros enfrente de ellos, los muros son impenetrables y los sensores están colocados uno al costado izquierdo de Karel, otro enfrente de Karel y el tercero en el costado derecho de Karel.

Cada sensor es activado cuando detecta enfrente del mismo un muro, así que, un sensor genera uno de dos valores dependiendo de una variable que pregunta si existe o no muro frente al sensor. Ejemplo, si la variable es “frente-libre” y el sensor detecta que no existe un muro enfrente, el sensor asigna el valor “verdadero” a la variable “frente-libre”

También Karel tiene un sensor que detecta cuando esta junto a un zumbador, de la misma forma que los anteriores sensores genera uno de dos valores verdadero si esta junto a zumbador o falso si no lo esta. La mochila de Karel también trabaja como sensor ya que puede haber zumbadores en ella o no. por ejemplo, si en la mochila hay 5 zumbadores, la variable “algún-zumbador-en-la-mochila” tendrá el valor de verdadero y falso en el caso de que no existieran zumbadores en la mochila.

Karel tiene un brazo mecánico que recoge zumbadores colocados en las esquinas de su mundo y coloca en la mochila. El brazo también puede tomar zumbadores de la mochila y colocarlos en cualquier esquina de su mundo. Este brazo mecánico no cuenta con ningún tipo de sensor, de manera que si le pedimos que recoja zumbadores de una esquina en particular y en esa esquina no existieran zumbadores, Karel generará un error grave y se apagará. Lo mismo pasaría si le pedimos que tome zumbadores de su mochila y la mochila estuviera vacía, por lo tanto es responsabilidad del usuario de Karel usar el brazo mecánico en situaciones que permitan su correcto funcionamiento.

1.3 Descripción del mundo de Karel

El mundo de Karel esta delimitado al Oeste por un muro vertical que se extiende hacia el Norte indefinidamente, hacia el sur esta limitado por un muro horizontal que se extiende hacia el Este indefinidamente. Estos muros son impenetrables por Karel y la intersección de estos dos muros se conoce con el nombre de Origen de Karel, siempre que se inicia una rutina de Karel, este aparecerá colocado en su origen de manera predeterminada a menos que se especifique lo contrario. El mundo de Karel esta dividido por avenidas verticales y calles horizontales que se extienden de manera indefinida en el mundo. La intersección de las calles y avenidas se conocen por esquinas.

El mundo de Karel también se complementa con otros objetos que llevan por nombre zumbadores, estos zumbadores se representan por numero colocados en las esquinas del mundo de Karel, cada esquina puede contener 0, 1, 2, ..., 99, 100,... o infinito número de zumbadores que pueden ser recogidos o colocados por Karel en la ejecución del algoritmo.

Estos elementos permiten diseñar una representación grafica de una situación en particular que presenta una problemática y de la cual tenemos que enseñarle a Karel a resolver.

NOTA IMPORTANTE: *Las instrucciones parecen tener errores de sintaxis, pero el programa no permite incluir acentos, por lo tanto aparecen sin acento de manera intencional.*

2 Comunicación con Karel

Una vez diseñado el mundo de Karel, comienza la tarea de enseñarle a desarrollarse en su mundo para dar una solución al problema que presenta el mundo. Al hablar de enseñarle a Karel estamos hablando de que debemos tener una comunicación con él a manera de que tenga las instrucciones necesarias para hacer una acción en particular. La manera en la que entablaremos esa comunicación es a través de un lenguaje de computación muy sencillo, este lenguaje, que es propio de Karel, esta basado en un lenguaje más sofisticado llamado Pascal.

2.1 Instrucciones Definidas en Karel

Adicional al lenguaje Pascal, Karel tiene cinco instrucciones definidas y concretas. El uso de estas instrucciones obliga a que el mundo de Karel cuente con las condiciones necesarias para su correcta ejecución, es decir, no podemos pedirle a Karel que avance si frente a el existe un muro, recordemos que los muros son objetos en el mundo de Karel que son impenetrables. Esta situación generaría un error grave y Karel terminaría su ejecución de manera inesperada.

Existen entonces 3 instrucciones definidas que generarían errores graves, estas instrucciones definidas, llamémoslas "instrucciones críticas" son las siguientes:

Instrucción Crítica	Acción	Casi crítico
avanza	Avanza una calle o avenida a la vez en dirección hacia la orientación actual de Karel.	Genera error grave si frente a Karel existe un muro al momento de su ejecución.
coge-zumbador	Esta instrucción activa el brazo mecánico de Karel y le indica que recoja un zumbador de la posición actual de Karel.	Genera error grave si junto a Karel no existe ningún zumbador para recoger.
deja-zumbador	Esta instrucción activa el brazo mecánico de Karel y le indica que tome un zumbador de su mochila y lo coloque en la posición actual de Karel.	Genera error grave si en la mochila de Karel no hay zumbadores.

Las otras dos instrucciones definidas, no producen errores graves, pero en el caso de ciclos de repetición podrían generar resultados inesperados en la ejecución del programa:

Instrucción Crítica	Acción
gira-izquierda	Provoca que Karel gire sobre su eje en un ángulo de 90° hacia la izquierda de Karel.
apagate	Esta instrucción le indica a Karel que se detenga apague sus sensores, su brazo mecánico y que finalice toda actividad.

Nótese que no existe una instrucción “encender”, “activar” o similar, pero si una instrucción dentro del cuerpo del programa que dice “Inicia-ejecución”, pero se da por un hecho que Karel esta siempre prendido a menos que se ejecute la instrucción “apagate”.

2.2 Sintaxis de instrucciones

Tenemos dos definiciones claves de la palabra Sintaxis:

En el ámbito de la gramática.- Parte de la gramática que estudia la forma en que se combinan y relacionan las palabras para formar secuencias mayores, cláusulas y oraciones y la función que desempeñan dentro de estas: la sintaxis estudia los tipos de oraciones.

En el ámbito de la informática.- Forma correcta en que deben estar dispuestos los símbolos que componen una instrucción ejecutable por el ordenador.

Para nuestro caso, la segunda definición es en la que basaremos el estudio de un lenguaje computacional. Un lenguaje computacional no es muy diferente de nuestro lenguaje que es un sistema de comunicación propio de una comunidad. Esta claro que la función de un lenguaje es ser un medio de comunicación con otras personas, en este mismo sentido, un lenguaje computacional es un medio de comunicación entre nosotros y la máquina, y como en todo sistema de comunicación existen ciertas normas y reglas que se deben de seguir para un correcto entendimiento entre las dos partes.

La sintaxis entonces es fundamental para que el ordenador, en este caso nuestro simulador de un robot, funcione correctamente. Por lo tanto, todas las instrucciones del lenguaje que maneja Karel deberán estar correctamente escritas.

“**inicia-programa**”
[Declaración de procedimiento”**;**”]
...
“**inicia-ejecución**”
Expresión General “**;**”
...
“**termina-ejecución**”
“**finalizar-programa**”

Declaración de un programa

En este ejemplo se muestra la sintaxis de la declaración de un programa en Karel. La sintaxis nos indica que las frases que están entre comillas y en negritas deben de ir escritas exactamente igual y ninguna de ellas debe de faltar. La frase entre corchetes cuadrados indica que en esa sección del programa se deben declarar los instrucciones que el usuario quiera definir, y los corchetes indican que una declaración de procedimiento es opcional, es decir, el usuario puede definir procedimientos o no hacerlo. La frase “Expresión General” indica que esta es la sección del programa donde escribiremos todas las instrucciones que deseamos que Karel ejecute, al no estar entre corchetes cuadrados la sintaxis indica que

Karel esperaría que en esa sección existan instrucciones, de lo contrario Karel no hará nada. Con esta sintaxis dada, veamos como quedaría un programa sencillo.

```
inicia-programa
        inicia-ejecución
                avanza;
        termina-ejecución
finalizar-programa
```

Programa 1

Nótese que al final de una "Expresión General" se coloca un punto y coma, el punto y coma son fundamentales para indicarle al programa que hemos terminado de escribir una instrucción completa. Existen instrucciones que se pueden escribir en más de una línea, sin embargo el punto y coma no debería ir uno por línea, sino, uno solo hasta el final de toda la instrucción.

```
si frente-libre entonces inicio
        avanza;
        Gira-izquierda;
fin;
```

Uso de punto y coma

En este ejemplo vemos tres punto y comas, avanza lleva punto y coma porque es una instrucción completa, según su sintaxis no hay más que escribir "avanza". Gira-izquierda también lleva su punto y coma porque es una instrucción completa, como en el caso de "avanza". El punto y coma que esta en la ultima línea corresponde a la instrucción "**si**" "**entonces"**. Este es un ejemplo de instrucciones que se escriben en más de una línea.

2.3 Instrucciones Definidas por el Usuario

Como vemos Karel esta limitado por estas cinco instrucciones y con ellas tenemos que enseñarle a Karel como resolver un problema dado. Estas instrucciones aunque pocas son suficientes para resolver un sin número de situaciones que puede encontrar en su mundo. Sin embargo podemos enseñarle a Karel nuevas instrucciones, estas nuevas instrucciones al no estar definidas por en el programa, tendremos que hacerlo nosotros.

Recordemos la sintaxis general de un programa en Karel:

```
"inicia-programa"
        [Declaración de procedimiento";"]
        ...
        "inicia-ejecución"
                Expresión General ";"
                ...
        "termina-ejecución"
"finalizar-programa"
```

Declaración de un programa

La declaración de nuevas instrucciones se hace en zona marcada cono "Declaración de procedimientos". Y como toda instrucción en Karel también tiene una sintaxis que se ha de seguir.

Sintaxis de una declaración de nueva instrucción (procedimiento):

"**define-nueva-instruccion**" Identificador ["("identificador")"] "**como**" "**inicio**" Expresión General";" "**fin**""**;**"

Declaración de una nueva instrucción

Donde:

- "identificador" es el nombre que le daremos a la nueva instrucción.
- Las frases que están entre comilla y en negritas deben de ir escritas exactamente igual y ninguna de ellas debe de faltar.
- Los corchetes indican que lo que esta dentro es opcional, es decir, el usuario puede usar parámetros o no hacerlo.
- La frase "Expresión general" indica que esta es la sección donde escribiremos todas las instrucciones que deseamos que Karel ejecute con esta nueva instrucción, al no estar entre corchetes cuadrados la sintaxis indica que Karel esperaría que en esa sección existan instrucciones, de lo contrario la nueva instrucción no hará nada.

Ejemplo:

"**define-nueva-instruccion**" gira-derecha **como inicio** gira-izquierda; gira-izquierda; gira-izquierda; **fin;**

Declaración de una nueva instrucción

Ahora estamos listos para usar una nueva instrucción que se llama gira-derecha y la forma de usarla es invocándola en el sección de "Expresión General" de nuestro programa principal.

inicia-programa "**define-nueva-instruccion**" gira-derecha **como inicio** gira-izquierda; gira-izquierda; gira-izquierda; **fin;** inicia-ejecución **gira-derecha;** termina-ejecución finalizar-programa

Programa 2

Una consideración importante al darle nombre a la nueva instrucción, es que esta, nunca debe de llevar el nombre de las instrucciones ya definidas en Karel, ni tampoco como las estructuras de control.

2.4 Funciones Booleanas.

Los sensores de Karel trabajan en conjunto con las funciones booleanas; son los sensores los que determinan el valor de cada función booleana. Las funciones se muestran en la siguiente tabla.

FUNCIÓN BOOLEANA		
De estado actual	De zumbadores	De orientación
frente-libre	junto-a-zumbador	orientado-al-norte
frente-bloqueado	no-junto-a-zumbador	orientado-al-sur
izquierda-libre	algun-zumbador-en-la mochila	orientado-al-este
izquierda-bloqueada	ningun-zumbador-en-la mochila	orientado-al-oeste
derecha-libre		no-orientado-al-norte
derecha-bloqueada		no-orientado-al-sur
		no-orientado-al-este
		no-orientado-al-oeste

Estas funciones booleanas solo pueden tener uno de dos valores, cierto o falso. Por lo tanto frente-libre puede tener el valor verdadero o falso dependiendo del resultado que genere el sensor que esta en la parte frontal de Karel. En las funciones booleanas al hablar de frente-libre, izquierda-libre o derecha-libre significa que no existe muro bloqueando la señal del sensor dado, es decir si frente-libre es verdadero, significa que el sensor de Karel no detecta muro frente a él, si existiera un muro, entonces frente-libre sería falso.

> Nota: Junto-a-zumbador significa que físicamente debe estar sobre zumbador y no como el caso de los muros que realmente están en frente de él, así, no-junto-a-zumbador significa que físicamente Karel no esta sobre un zumbador.

A continuación se presentan las funciones booleanas, su valor y el estado que representa cada una de ellas en la lógica de la ejecución de cualquier programa elaborado.

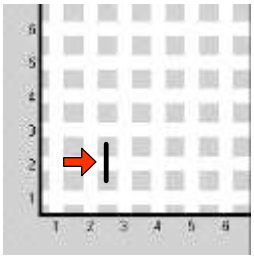
Función Booleana de Estado Actual	Valor	Estado físico de Karel
frente-libre	Verdadero	Si el sensor no detecta muro enfrente de él.
frente-libre	Falso	Si el sensor detecta muro enfrente de él.
frente-bloqueado	Verdadero	Si el sensor detecta muro enfrente de él.
frente-bloqueado	Falso	Si el sensor no detecta muro enfrente de él.
izquierda-libre	Verdadero	Si el sensor no detecta muro enfrente de él.

izquierda-libre	Falso	Si el sensor detecta muro enfrente de él.
izquierda-bloqueada	Verdadero	Si el sensor detecta muro enfrente de él.
izquierda-bloqueada	Falso	Si el sensor no detecta muro enfrente de él.
derecha-libre	Verdadero	Si el sensor no detecta muro enfrente de él.
derecha-libre	Falso	Si el sensor detecta muro enfrente de él.
derecha-bloqueada	Verdadero	Si el sensor detecta muro enfrente de él.
derecha-bloqueada	Falso	Si el sensor no detecta muro enfrente de él.

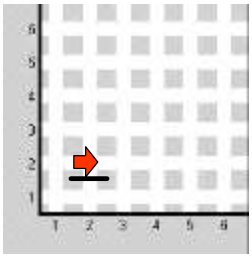
Función Booleana de Zumbadores	Valor	Estado físico de Karel
junto-a-zumbador	Verdadero	Si Karel esta sobre un zumbador.
junto-a-zumbador	Falso	Si Karel no esta sobre un zumbador.
no-junto-a-zumbador	Verdadero	Si Karel no esta sobre un zumbador.
no-junto-a-zumbador	Falso	Si Karel esta sobre un zumbador.
algun-zumbador-en-la mochila	Verdadero	Si Karel lleva zumbadores en la mochila.
algun-zumbador-en-la mochila	Falso	Si Karel no lleva zumbadores en la mochila.
ningun-zumbador-en-la mochila	Verdadero	Si Karel no lleva zumbadores en la mochila.
ningun-zumbador-en-la mochila	Falso	Si Karel lleva zumbadores en la mochila.

Función Booleana de Orientación	Valor	Estado físico de Karel
orientado-al-norte	Verdadero	Si Karel esta orientado al norte.
orientado-al-norte	Falso	Si Karel no esta orientado al norte.
orientado-al-sur	Verdadero	Si Karel esta orientado al sur.
orientado-al-sur	Falso	Si Karel no esta orientado al sur.
orientado-al-este	Verdadero	Si Karel esta orientado al este.
orientado-al-este	Falso	Si Karel no esta orientado al este.
orientado-al-oeste	Verdadero	Si Karel esta orientado al oeste.
orientado-al-oeste	Falso	Si Karel no esta orientado al oeste.
no-orientado-al-norte	Verdadero	Si Karel no esta orientado al norte.
no-orientado-al-norte	Falso	Si Karel esta orientado al norte.
no-orientado-al-sur	Verdadero	Si Karel no esta orientado al sur.
no-orientado-al-sur	Falso	Si Karel esta orientado al sur.
no-orientado-al-este	Verdadero	Si Karel no esta orientado al este.
no-orientado-al-este	Falso	Si Karel esta orientado al este.
no-orientado-al-oeste	Verdadero	Si Karel no esta orientado al oeste.
no-orientado-al-oeste	Falso	Si Karel esta orientado al oeste.

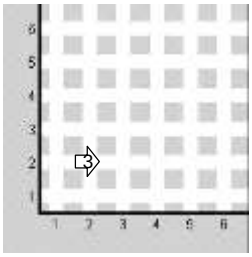
Ejemplos:



Función	Valor
frente-libre	Falso
frente-bloqueado	Verdadero



Función	Valor
derecha-libre	Falso
derecha-bloqueado	Verdadero



Función	Valor
junto-a-zumbador	Verdadero
no-junto-a-zumbador	Falso

3 Mensajes de error

No existe programador alguno que no cometa algún tipo de error, de hecho es común cometerlos, y necesitamos identificar cada error para poder corregirlo. Existen dos principales tipos de error, los errores lógicos y los errores de sintaxis.

3.1 Errores de Sintaxis

Los errores de sintaxis son muy fáciles de identificar, es Karel quien lo hace y nos avisa de que ha localizado un error de este tipo. Para esto, tenemos que decirle a Karel que **compile** el programa, compilar no es otra cosa que verificar que el programa este correctamente escrito en sintaxis y estructura.

En seguida listaremos una serie de errores que común mente cometemos, su significado y la posible solución.

CÓDIGO	ERROR	DESCRIPCIÓN
iniciar-programa inicia-ejecucion avanzar; apagate; termina-ejecucion		Este error es muy común. Se presenta cuando escribimos una instrucción que no existe o que no ha sido definida. En este caso se escribió **avanzar** en lugar de

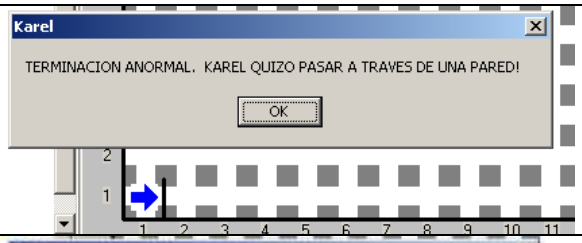
finalizar-programa		**avanza** que si es una instrucción definida.
iniciar-programa inicia-ejecucion avanza apagate; termina-ejecucion finalizar-programa		Este error es de los que más se cometen. Sucede cuando omitimos el indicador de fin de instrucción, es decir, el **punto y coma** al final de la instrucción. En este caso, omitimos el punto y coma después de la instrucción **avanza**.
iniciar-programa inicia-ejecucion repetir 3 veces inicio avanza; apagate; termina-ejecucion finalizar-programa		Otro error común es olvidarnos de colocar un **fin** por cada inicio que pongamos. En este código en la línea 3 colocamos un **inicio**, pero omitimos su correspondiente **fin** en las líneas sucesivas de código en el lugar donde corresponda según la lógica del mismo.
iniciar-programa inicia-ejecucion si frente libre entonces avanza; apagate; termina-ejecucion finalizar-programa		Error de sintaxis, en la línea 3 se escribió de forma errónea la función booleana frente-libre. El error consiste en omitir el guión entre las dos palabras.
iniciar-programa inicia-ejecucion si frente-libre entonces avanza; sino gira-izquierda; apagate; termina-ejecucion finalizar-programa		La palabra **sino** no es por si sola una instrucción definida, sino que es parte de otra instrucción más compleja que es: **si** Termino **entonces** expresión1 **sino** expresión2 y el punto y coma de la línea 4 da por terminada la instrucción **si entonces**. Se requiere de eliminar el punto y coma después del avanza.
iniciar-programa inicia-ejecucion si frente-libre hacer avanza; apagate; termina-ejecucion finalizar-programa		Recordemos que la sintaxis de la instrucción es **si** termino **entonces** y no **si** termino **hacer**.

Estos fueron algunos de los errores que comúnmente cometemos al codificar un algoritmo en Karel, no son todos por supuesto, pero lo que tenemos que tener bien presente es que si estamos en la sección de programa y compilamos y nos marca un error, este siempre será un error de sintaxis y su corrección es tan simple, como ir a la pestaña de ayuda y ver la forma en la que se debe escribir correctamente la instrucción, es decir, revisar su sintaxis.

3.2 Errores de lógica.

Estos errores se cometen después de corregidos los de sintaxis, y los detecta Karel en la ejecución del programa. Por lo tanto, después de compilar el programa y de corregir todos los errores de sintaxis que se presentaron pasamos a la pestaña **ejecutar** en donde, después de **inicializar** el programa al ejecutarlo, se presenta un error grave y la ejecución se detiene. Estos errores se corrigen, modificando la lógica del programa, es decir, si queremos que Karel avance tenemos que ver si el mudo de Karel cuenta con las condiciones necesarias para que Karel avance libremente, por lo tanto, para que pueda avanzar sin ningún problema, Karel deberá tener su frente libre.

Veamos algunos de los errores típicos que se comenten el la lógica de la ejecución de un programa en Karel.

CÓDIGO	ERROR	DESCRIPCIÓN
iniciar-programa inicia-ejecucion avanza; apagate; termina-ejecucion finalizar-programa		Error clásico en la ejecución de un programa, se le esta pidiendo a Karel que avance, pero en frente de el existe un muro, como Karel no puede atravesar muros, el programa es detenido y avisa de que se produjo un error grave.
iniciar-programa inicia-ejecucion coge-zumbador; apagate; termina-ejecucion finalizar-programa		Este error se genera cuando se le ordena a Karel recoger zumbadores de su mundo y sin embargo no existe ningún zumbador junto a Karel. Recordemos que para que Karel recoja un zumbador debe estar sobre el zumbador aunque la instrucción diga junto-a-zumbador.
iniciar-programa inicia-ejecucion deja-zumbador; apagate; termina-ejecucion finalizar-programa		Si en las condiciones iniciales del mundo de Karel o en la misma ejecución no cuidamos de que tenga zumbadores en la mochila, al ejecutar la instrucción deja-zumbador nos marcará el error mostrado.

Existen otros errores lógicos que aunque no son marcados por Karel, es evidente que existen. Uno de ellos es cuando le pedimos a Karel que haga una acción y no consideramos cuando se detenga en la ejecución de la misma, por ejemplo, si le pedimos a Karel que avance mientras su frente este libre, debemos de tener en cuanta que debe haber una condición que haga que Karel se detenga en algún momento, ya que de no hacerlos, Karel podría avanzar indefinidamente.

4 Estructuras de control e iteración.

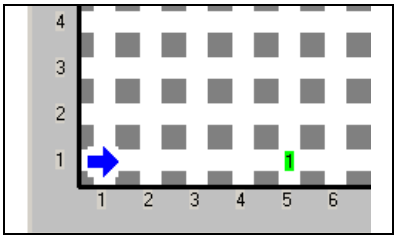
Existen dos tipos de estructuras de programación en el lenguaje que maneja Karel. Son las estructuras de control y las estructuras de iteración. Estas estructuras son fundamentales en la lógica de un programa. Nos permiten decidir entre realizar una acción o grupo de acciones y realizar otras diferentes. También podemos repetir un número finito de veces una acción o un grupo de acciones, así como repetir mientras una condición se cumpla.

4.1 Estructuras de iteración.

Para realizar iteraciones tenemos dos estructuras bien definidas en Karel. Estas estructuras repiten un bloque de instrucciones un número determinado de veces siempre que se cumpla una condición o que indiquemos el número exacto de las iteraciones.

4.1.1 La instrucción Repetir.

La instrucción repetir hace un número definido de iteraciones. Supongamos que queremos que Karel llegue hasta el zumbador como se muestra en la figura.



Una posible solución sería la siguiente:

```
iniciar-programa
    inicia-ejecucion
        avanza;
        avanza;
        avanza;
        avanza;
        apagate;
    termina-ejecucion
finalizar-programa
```

Pero imaginemos que el zumbador se encuentra no en la avenida 5 sino en la 20, 30, 50 o 150, sería muy engorroso escribir avanza tantas veces como se necesite. Esto se resuelve con la instrucción de control **Repetir**. Su sintaxis es la siguiente:

```
"repetir" ExpresionEntera "veces" "inicio"
        expresión;
"fin;"
```

Sintaxis de **Repetir**

Donde

- ExpresiónEntera es un valor numérico.
- Y expresión es una instrucción o grupo de instrucciones.

Y aplicada al ejemplo tendríamos

```
iniciar-programa
   inicia-ejecucion
      repetir 4 veces inicio
         avanza;
      fin;
      apagate;
   termina-ejecucion
finalizar-programa
```

Que nos da el mismo resultado, pero si se exactamente donde esta el zumbador y cuantas repeticiones de la instrucción **avanza** necesito para llegar a él, solo cambió el valor de la ExpresiónEntera por el valor correcto y me habré ahorrado el escribir tanto código como en el caso anterior, o en los casos donde el zumbador estuviera a una distancia más grande.

4.1.2 La instrucción Mientras.

¿Pero que tal si no se en que avenida se encuentra realmente el zumbador? ¿y si la posición del zumbador cambia para cada ejecución que haga del programa?. La respuesta es que no tendré un número de avances definido, para cubrir esta situación existe la instrucción **mientras**, y trabaja bajo la siguiente sintaxis.

```
"mientras" Termino "veces" "inicio"
        expresión;
"fin;"
```

Sintaxis de **Mientras**

Donde

- Termino es una función booleana.
- Y expresión es una instrucción o grupo de instrucciones.

Si usamos el ejemplo anterior, la solución del problema quedaría:

```
iniciar-programa
   inicia-ejecucion
      mientras no-junto-a-zumbador hacer inicio
         avanza;
      fin;
      apagate;
   termina-ejecucion
finalizar-programa
```

Que también nos genera el mismo resultado pero este código es más general, es decir, funciona para cualquier otro mundo con un zumbador en cualquier avenida sobre la calle 1.

4.2 Estructura de control.

Karel cuenta con una sola instrucción de control, también es conocida como Estructura de Control de Flujo de Programa. Recibe este nombre por ser una estructura que controla, como su nombre lo indica, el flujo del programa, es decir, que usando una función booleana controla la ejecución de un bloque de código en particular u otro bloque diferente. Como recordaremos la función booleana solo puede ser Falso o Verdadero, entonces, dependiendo de el valor de la función la estructura de control ejecutará una instrucción o bloque de instrucciones o un bloque de código o instrucción diferente.

4.2.1 Instrucción Si Entonces.

La instrucción s**i entonces** ejecuta un bloque de código dependiendo del valor de una función booleana, quiere decir que el bloque de código puede ser ejecutado o no, esto depende del valor de la función booleana, por ejemplo depende del valor de **frente-libre** para que Karel haga un bloque de código o no lo haga.

Sintaxis de la instrucción **si entonces**:

```
"si" Termino "entonces" "inicio"
        expresión;
"fin;"
```

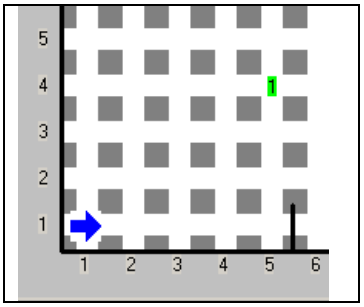
Sintaxis de **si entonces**

Donde:

- Termino es una función booleana.
- Y expresión es una instrucción o grupo de instrucciones.

Ejemplo:

Supongamos que queremos que Karel avance hasta llegar al zumbador, como en el problema anterior, solo que el mundo ahora es el siguiente:



Si usamos el código anterior tendríamos:

iniciar-programa
  inicia-ejecucion
  mientras no-junto-a-zumbador hacer inicio
  avanza;
  fin;
  apagate;
  termina-ejecucion
Finalizar-programa
	

Según el mundo, necesitamos que cuando Karel detecte su frente bloqueado, haga un giro a la izquierda para esquivar el muro y continuar avanzando hasta el zumbador. Entonces, necesitamos que Karel haga una acción (girar a la izquierda) si se cumple con la función booleana es decir, cuando esta sea Verdadera.

si frente-bloqueado **entonces inicio**   Gira-izquierda; **fin;**
Instrucción **si entonces**

El código final sería:

iniciar-programa
  inicia-ejecucion
  mientras no-junto-a-zumbador hacer inicio
  si frente-bloqueado **entonces inicio**
  gira-izquierda;
  fin;
  avanza;
  fin;
  apagate;
  termina-ejecucion
finalizar-programa
	

4.2.2 Instrucción Si Entonces Sino.

Esta instrucción realmente no es otra instrucción diferente a **si entonces**, sino que es realidad es la instrucción completa. La versatilidad de esta instrucción nos permite considerarla como dos instrucciones distintas aunque no lo sean. Veámoslo de otra manera, la instrucción **si entonces sino**, no permite hacer un bloque de código si se cumple una condición (función booleana), o nos permite hacer uno de dos bloques de código si se cumple o no una condición.

Esto es, sean los bloques de código A y B.

Instrucción	Se ejecuta si
si frente-libre entonces inicio A; fin;	El código A se ejecuta o no, dependiendo de si el frente este libre o no.
si frente-libre entonces inicio A; fin sino inicio B; fin;	Se ejecuta el código A si el frente esta libre y si no lo esta, se ejecuta el bloque de código B.

Si usamos esta modalidad de la instrucción **si entonces sino**, nuestro código del problema anterior lo podemos escribir así:

```
iniciar-programa
   inicia-ejecucion
      mientras no-junto-a-zumbador hacer inicio
         si frente-bloqueado entonces inicio
            gira-izquierda;
         fin
         sino inicio
            avanza;
         fin;
      fin;
      apagate;
   termina-ejecucion
finalizar-programa
```

Como vemos obtenemos el mismo resultado, pero nuestro código esta escrito de una manera más estructurada y se puede leer mejor.

Por último, la sintaxis general de esta instrucción es:

```
si Termino entonces inicio
        Expresión1;
fin
[sino inicio
        Expresión2;
fin];
```

Sintaxis general de la Instrucción **si entonces sino**

4.3 Las funciones sucede y precede

Ahora ya sabemos cómo mandarle un número a un procedimiento, y probablemente ya habrás intentado poner operaciones como suma, resta o multiplicación, sin embargo lamanto decirte que ninguna de estas operaciones están soportadas en Karel.

Por otro lado, existen dos funciones que nos permiten sumarle 1 a un número y restarle 1.

Pero... ¿qué es una función? Una función es una instrucción que devuelve un valor, es decir, reciben un parámetro (o más) que luego procesa, para al final regresar un valor; por ejemplo, la función booleana junto-a-zumbador devuelve verdadero si Karel está parado junto a un zumbador y falso si no lo está. En Karel no se pueden declarar funciones nuevas, pero se pueden usar las que ya existen.

Las funciones sucede y precede son dos instrucciones que reciben un parámetro, posteriormente, devuelven un número más y un número menos (respectivamente) que el que le enviamos.

La función sucede se escribe así:

```
sucede(xxx);
donde xxx es un número o un parámetro, y la función precede se escribe así:
precede(xxx);
donde xxx es un número o un parámetro.
```

Debido a que devuelven un número, solo nos pueden servir poniéndolas en alguna instrucción o sentencia que reciba un número, como repetir/veces, otro sucede o precede o una instrucción personal que reciba un parámetro.

Por ejemplo, el siguiente trozo de código pone n + 1 zumbadores en donde Karel se encuentra:

```
repetir sucede(n) veces inicio
  deja-zumbador;
fin;
```

nota que n se "incrementa" (se le suma uno). Si en vez de sucede, pusiéramos precede, Karel dejaría n - 1 zumbadores, porque la n se "decrementa" (se le quita uno) cuando se pone dentro de una función precede.

4.4 La función si-es-cero

La última función en Karel, es la función si-es-cero, que nos ayuda a saber si un número es cero. Devuelve verdadero si el número es cero y falso si no lo es.

Es evidente de que si ponemos "si-es-cero(0)" no es muy útil, ya que sabemos perfectamente que cero es cero (:S). Sin embargo es muy útil cuando se está manejando parámetros. Por ejemplo, queremos hacer una instrucción que avance "n" lugares, pero si el parámetro es cero, gire a la izquierda.

Por razones didácticas, en esta ocasión te daremos la solución:

define-nueva-instruccion avanza-si-no-es-cero (n) como inicio

```
  si si-es-cero (n) entonces inicio
    gira-izquierda;
  fin;
  sino inicio
    repetir n veces inicio
      avanza;
    fin;
  fin;
fin;
```

5. Ejemplo de cómo resolver un problema

Una vez que has comprendido la importancia de contar con una metodología para hacer programas, te mostraré un ejemplo de cómo los debes resolver. Recuerda que esto es un entrenamiento, ya que debes cambiar los hábitos en la manera de hacer los programas.

Preparemos nuestras herramientas de trabajo: 3 hojas en blanco y nuestro lápiz bien afilado. A continuación veremos el problema que debes de resolver y para el cual debemos llegar a resolverlo en todos los casos, ó sea que nos dé 100 puntos. Sin más preámbulo, vamos a leerlo.

5.1. Problema: Los canales del lago

Descripción

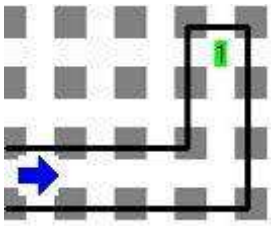
Como es bien sabido por todos, el lago de Xochimilco es un hermoso lago que año con año, y desde hace muchísimo tiempo, atrae una gran cantidad de turistas, con el propósito de poder dar un viaje en el lago en una de las trajineras.
Durante el recorrido, uno puede comprar comida, contratar a los mariachis, y disfrutar del paisaje; todo eso sobre las barcazas.
El viaje en trajinera consiste en recorrer los canales que han sido construidos sobre del lago, empezando en un lugar y terminando en otro.

Problema

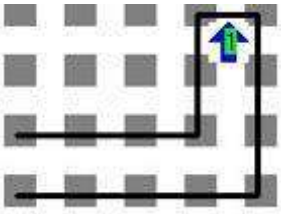
El día de hoy Karel va a visitar el lago, y tú, como dueño de la trajinera Morpheus, tienes que llevarlo por el recorrido.

Consideraciones

- Karel inicia al principio del recorrido orientado hacia donde éste se dirige.
- Karel no tiene zumbadores en la mochila.
- Los canales del lago son muy angostos, y se representan con un camino de ancho de 1.
- Mientras vas en el recorrido, recuérdale a Karel no tirar basura en el lago.
- Puedes saber que llegaste al final del recorrido, porque encontrarás un beeper.
- El camino sólo se cierra al final del recorrido, junto al beeper.
- Karel debe de terminar en el final del recorrido.

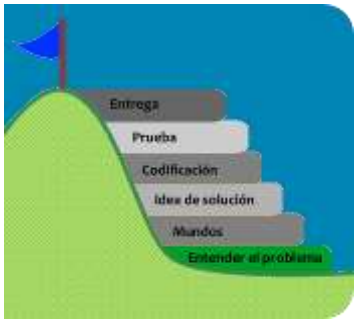


Mundo de ejemplo

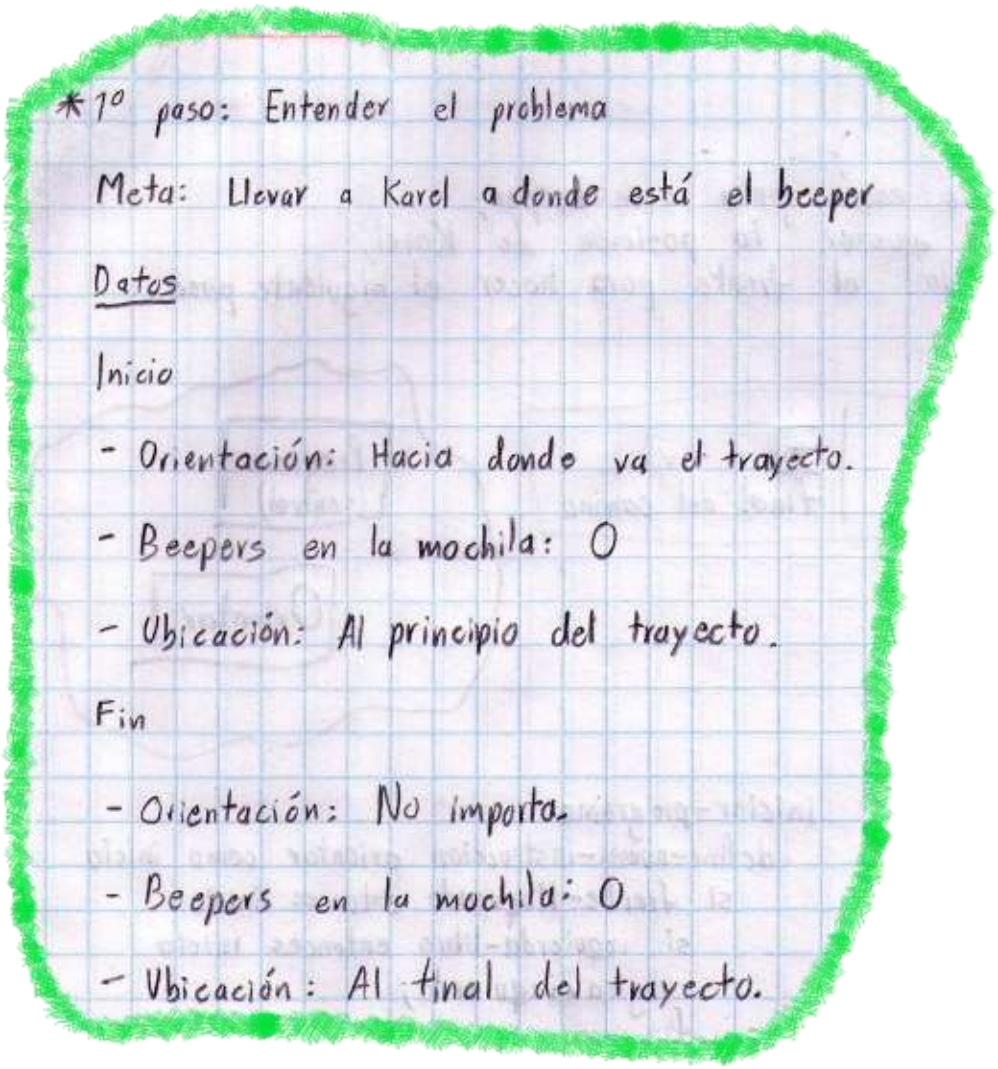


Solución al mundo de ejemplo

5.2. 1er. Paso: Entender el problema

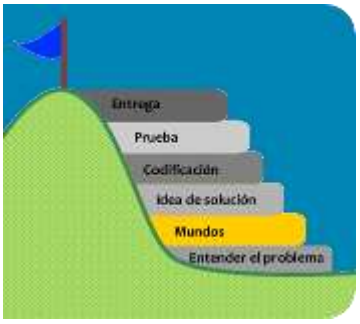


Lo primero que debemos hacer es analizar nuestro problema; para esto lo debemos leer hasta que nos quede claro, si queda alguna duda, la debemos preguntar.
Para saber que ya lo entendimos, debemos ser capaces de explicarlo a otra persona y a nosotros mismos. Una manera de saber que estamos bien es sacar los datos del problema de manera similar a como si fuera un problema de matemáticas y anotarlos en nuestra hoja; estos datos serían: zumbadores en la mochila, orientación inicial de Karel, límites del problema, orientación final de Karel, etc. En la siguiente figura se ve un ejemplo para este problema:



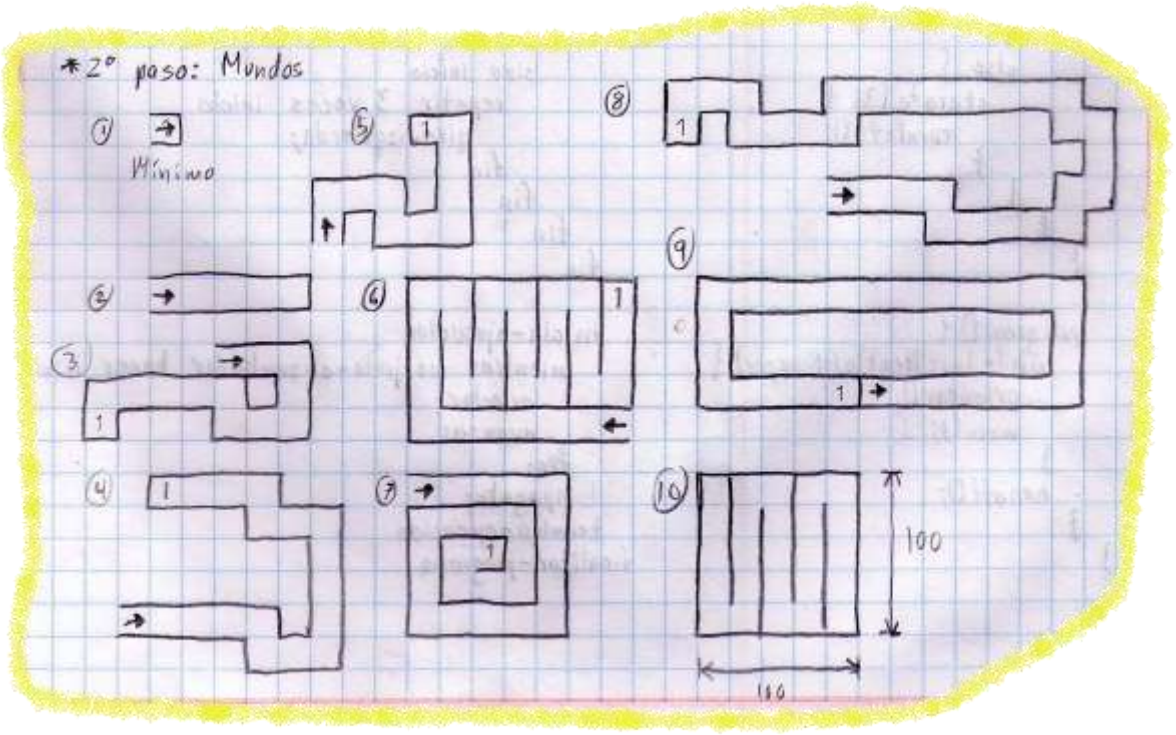
¿Esto me acerca a la solución, por qué?

5.3. 2° Paso: Mundos



Una vez que tenemos una idea más clara de los que nos piden, es bueno pensar en los diferentes mundos a los que aplica la descripción del problema (te recuerdo que realizar esto cuenta entre un 20% y un 40% de tu calificación final), ya que como se mencionó anteriormente, si podemos pensar en las diferentes posibilidades con que nos van a evaluar, seguramente nos llevará a pensar una solución que satisfaga el 100% de los mundos con que nos calificarán.

En la siguiente figura podemos apreciar que fuimos capaces de encontrar 10 casos diferentes y si hacemos un programa que sea capaz de resolver estos 10 casos, ten la seguridad de que tendrás una alta calificación.



Revisando los casos, puedes ver que no es necesario dibujar perfectamente el mundo de Karel, sino que debemos escribirlos de una forma clara, sencilla y rápida. Por ejemplo, en el caso 10 que mide 100x100, se indica el ancho y largo de 100 con unas señales de acotamiento similares a las que se usan en planos.

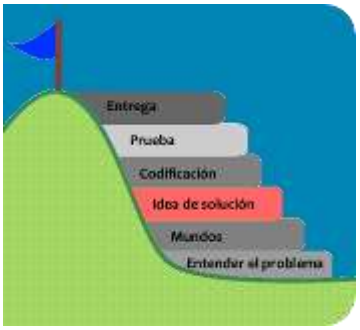
A continuación haremos algunas observaciones de los casos desarrollados:

- ✓ **Caso 1**: Es el caso mínimo, ya que no existe un caso más pequeño que este.
- ✓ **Caso 8**: Muestra un caso en el que se tiene que ajustar el camino de Karel en las 4 direcciones posibles: norte, sur, este y oeste. Este caso lo podríamos considerar como un caso máximo en cuanto a las direcciones que debe de tomar Karel para llegar a su objetivo (en este caso llegar al final del canal).
- ✓ **Caso 10**: Muestra un caso máximo en cuanto a longitud del recorrido, ya que debe atravesar todo el mundo (si te da tiempo, constrúyelo; si no, trata de hacer algo que lo iguale en condiciones y características).
- ✓ **Otros casos**: Los podemos considerar como casos típicos o promedio. Siempre en bueno hacer 3 o 4 casos promedio.

Te aseguro que si tienes la suficiente paciencia y dedicación para trabajar de esta forma de ahora en adelante en cada unos de tus problemas de programación, el resultado que obtendrás te llevará a consolidarte como un verdadero ganador.

¿Esto me acerca a la solución, por qué?

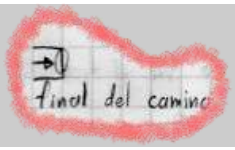
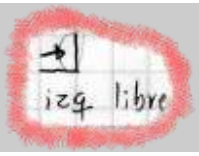
5.4. 3er. Paso: Idea de solución



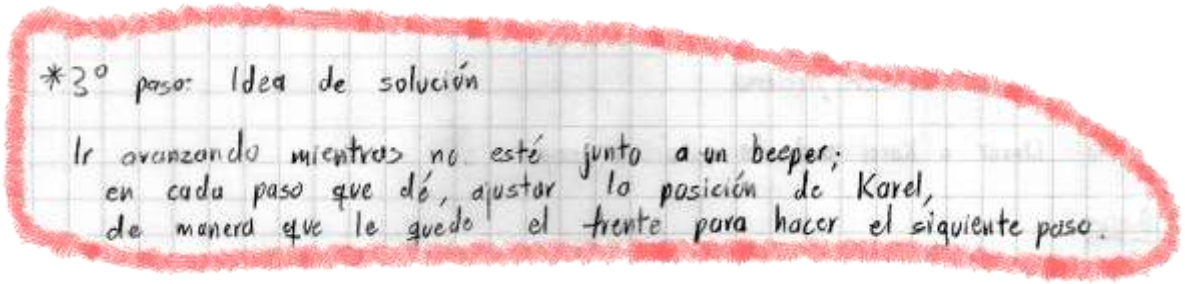
Una vez que ya hemos comprendido el problema, debemos pensar en alguna solución (más adelante se hablará más de cómo ayudarse a tener ideas de solución). De una manera simple nos podemos basar en las siguientes 3 ideas:

"Que propiedades tiene este problema", "Otra forma de plantearlo", "subdividir en casos", "se parece a algo ya resuelto".

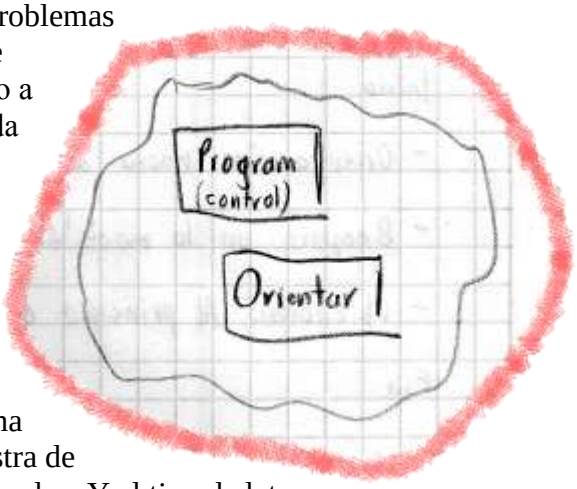
En este caso propongo **"otra forma de plantearlo"** y **"que propiedades tiene este problema"** debido a que podemos ver el problema como hacer que Karel camine hasta que encuentre un zumbador. La idea es hacer un programa que haga que Karel camine mientras no esté junto a un zumbador. Dentro de las propiedades de este problema destaca el que Karel no debe regresar por un camino que ya ando, si observamos bien a Karel solo podrá caminar hacia adelante, girar a la izquierda o girar a la derecha, basados en esta propiedad del problema cuando el camino presente algunos quiebres (variaciones de orientación hacia donde caminará Karel), Bastaría con que Karel ajuste su orientación ya sea a la izquierda o la derecha (dependiendo de donde tenga el camino libre) y hacer que Karel avance una vez que se ha orientado correctamente (que el frente este libre), esto es, se debe ajustar la orientación de Karel con la finalidad de lograr una posición en que pueda moverse (de ser posible, una posición de frente libre) y avanzar. De los casos que construimos, podemos apreciar que cada vez que avanza Karel pueden existir las siguientes 4 situaciones:

1. **Si está junto a un zumbador, ahí termina. No debemos hacer nada porque ahí termina el programa.**	
2. Que el frente esté libre. Para este caso no debemos hacer ningún ajuste.	
3. Que el frente esté bloqueado y la izquierda libre. En este caso debemos ajustar la orientación de Karel haciéndolo girar una vez a la izquierda.	
4. Que el frente esté bloqueado y la derecha libre. En este caso debemos ajustar la orientación de Karel haciéndolo girar a la derecha (esto se hace con 3 giros a la izquierda de Karel).	

Con lo que vemos, el problema se resume a un problema sencillo de hacer: que Karel camine y ajuste su orientación mientras no esté junto a un zumbador.
Es importante que te des cuenta que todo esto se hace en la mente y, ayudado de nuestras herramientas exclusivas (papel y lápiz); nosotros debemos anotar la idea para que no se nos olvide como se muestra a continuación:



Nuestro programa se puede ver muy simple. Nos ayudaremos de un diagrama de bloques (llamado de bloques porque usamos cuadros para representar las ideas generales). Como podemos apreciar, la nube que rodea nuestros bloques representa el problema que una vez habiéndolo dividido en 2, resulta más simple resolver 2 problemas menores que uno mayor. Esta figura muestra el bloque de control que hará que "Karel camine mientras no está junto a un zumbador", y el bloque de la nueva instrucción llamada orientar que "ajustará la orientación de Karel en caso de ser necesaria".



En este momento, ya estamos listos para planificar el problema de acuerdo a la idea de la sección anterior, es decir, construir la escalera para llegar nuestra meta que se ubica en la cima. Esto no lo debemos hacer; sin embargo, en este apunte lo mostraré para ver cómo se unen los conceptos teóricos de elaboración de un programa con la práctica de hacer un programa. Como sólo se muestra de una manera ilustrativa, utilizaremos los recuadros redondeados. Y el tipo de letra en cursivas.

¿Esto me acerca a la solución, por qué?

5.5. 4° Paso: Codificación

Ahora lo que debemos hacer es desarrollar el programa de control y cada una de las nuevas instrucciones que vamos a necesitar. En este caso son el *programa principal* que tiene la rutina de control y la nueva instrucción *orientar*.

Las siguientes figuras muestran el desarrollo en papel de las rutinas más importantes, ya que las más pequeñas y sencillas se pueden hacer directamente en el teclado cuando ya se tiene algo de experiencia. Mientras adquieres la experiencia, te aconsejo que trates de escribirlas todas en el papel antes de escribirlas en el Karel.

El programa principal quedaría de la siguiente manera en Java y/o Pascal; si lo observamos sigue nuestra idea: mientras Karel no esté junto a un zumbador, avanza y se orienta.

```
program() {
  while (notNextToABeeper) {
    orientar();
    move();
  }
  turnoff();
}
```

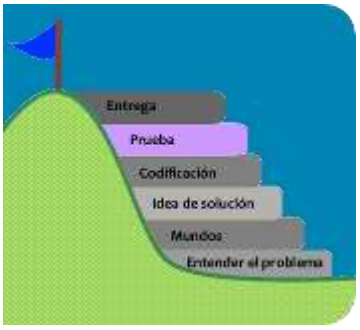
```
inicia-ejecucion
  mientras no-junto-a-zumbador hacer inicio
    orientar;
    avanza;
  fin;
  apagate;
termina-ejecucion
```

En la parte de la orientación quedaría como lo muestra la siguiente figura. Una buena práctica es hacerle una prueba de escritorio y ver que funciona en todos los casos.

```
program {
void orientar() {
  if (frontIsBlocked) {
    if (leftIsClear) {
      turnleft();
    }
    else {
      iterate (3) {
        turnleft();
      }
    }
  }
```

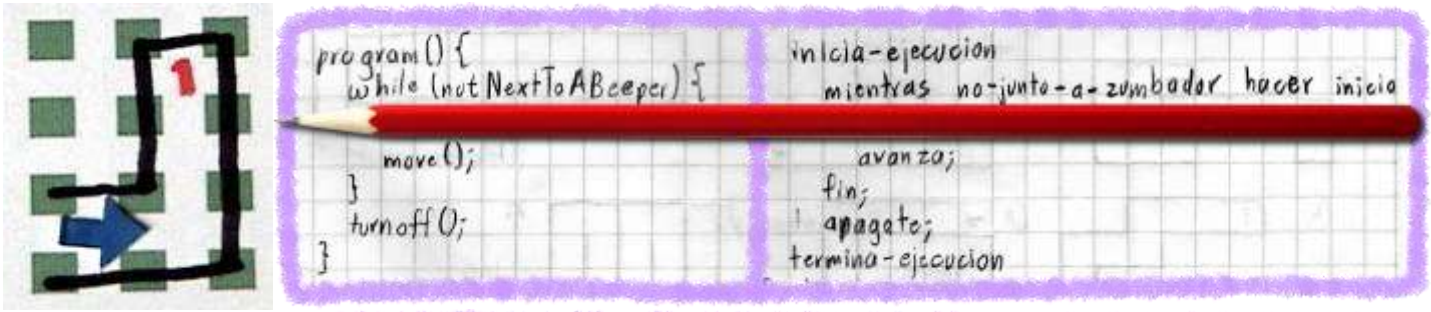
```
define-nueva-instruccion orientar como inicio
  si frente-bloqueado entonces inicio
    si izquierda-libre entonces inicio
      gira-izquierda;
    fin
    sino inicio
      repetir 3 veces inicio
        gira-izquierda;
      fin
    fin
  fin
fin
```

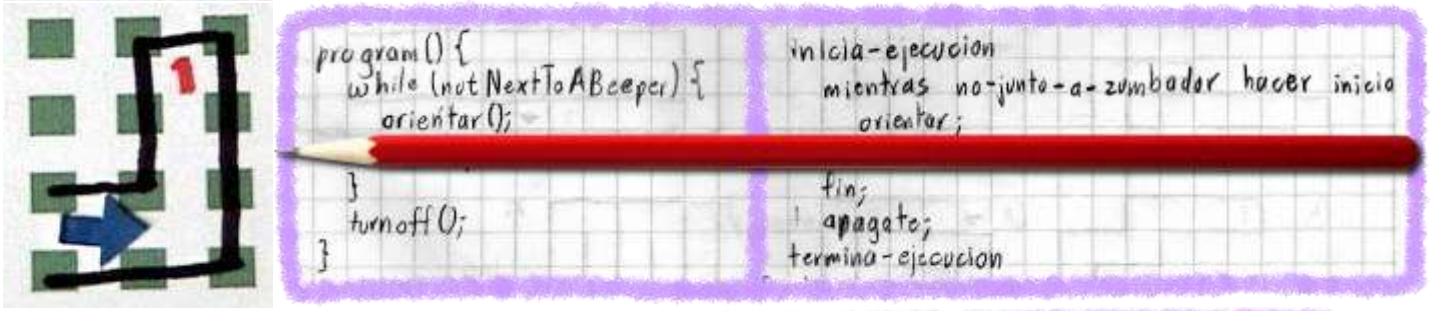
5.6. 5° Paso: Prueba

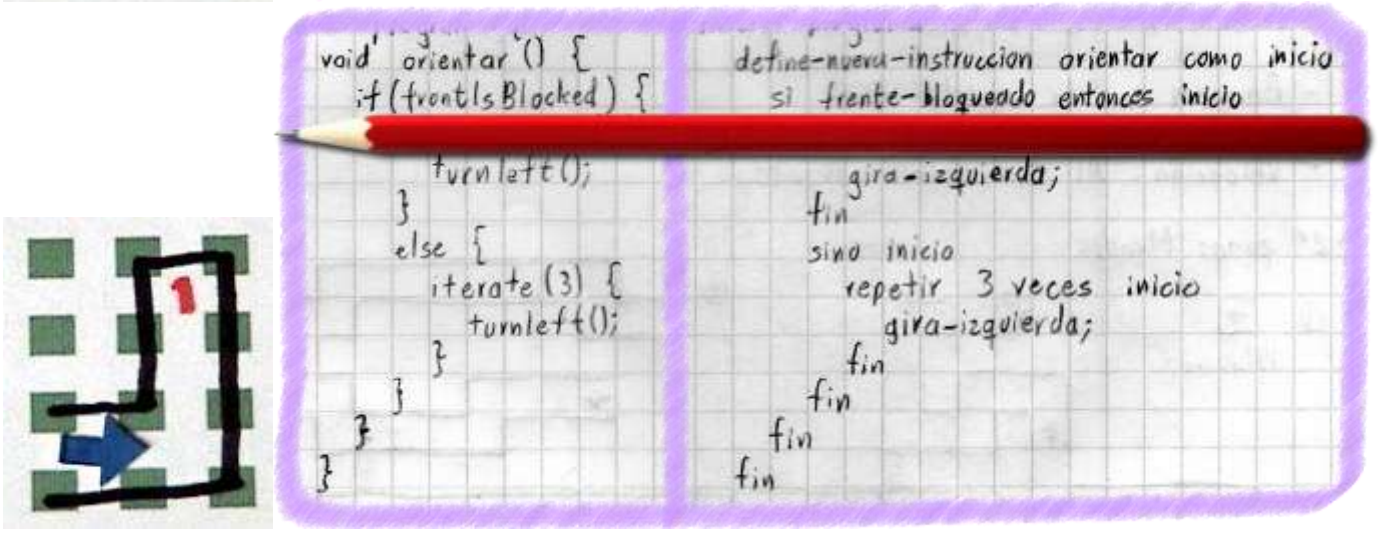


Haremos la prueba de escritorio (recuerda que se llama así porque la hacemos en el escritorio de trabajo utilizando un Karel y nuestras hojas de papel).

Haremos esta prueba para el caso de que Karel deba ajustar su orientación a la izquierda como se muestra a continuación. Nuestra rutina de control haría que Karel comprobara su posición y caminara un paso:

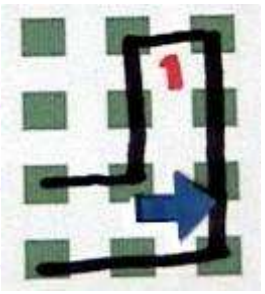




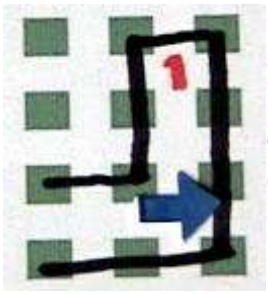


```
program () {
  while (not NextToABeeper) {
    orientar();
    move();
  }
  turnoff();
}
```

```
inicia-ejecucion
  mientras no-junto-a-zumbador hacer inicio
    orientar;
    avanzo;
  fin;
  apagate;
termina-ejecucion
```

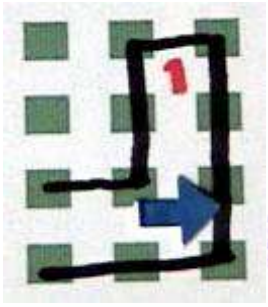


Dejándonos en esta posición, se llamará a la rutina *orientar*, la cual pregunta si el frente está bloqueado, cumpliéndose en este caso: por lo tanto, pregunta si la izquierda está libre, siendo afirmativamente, haciendo que Karel haga un giro a la izquierda.



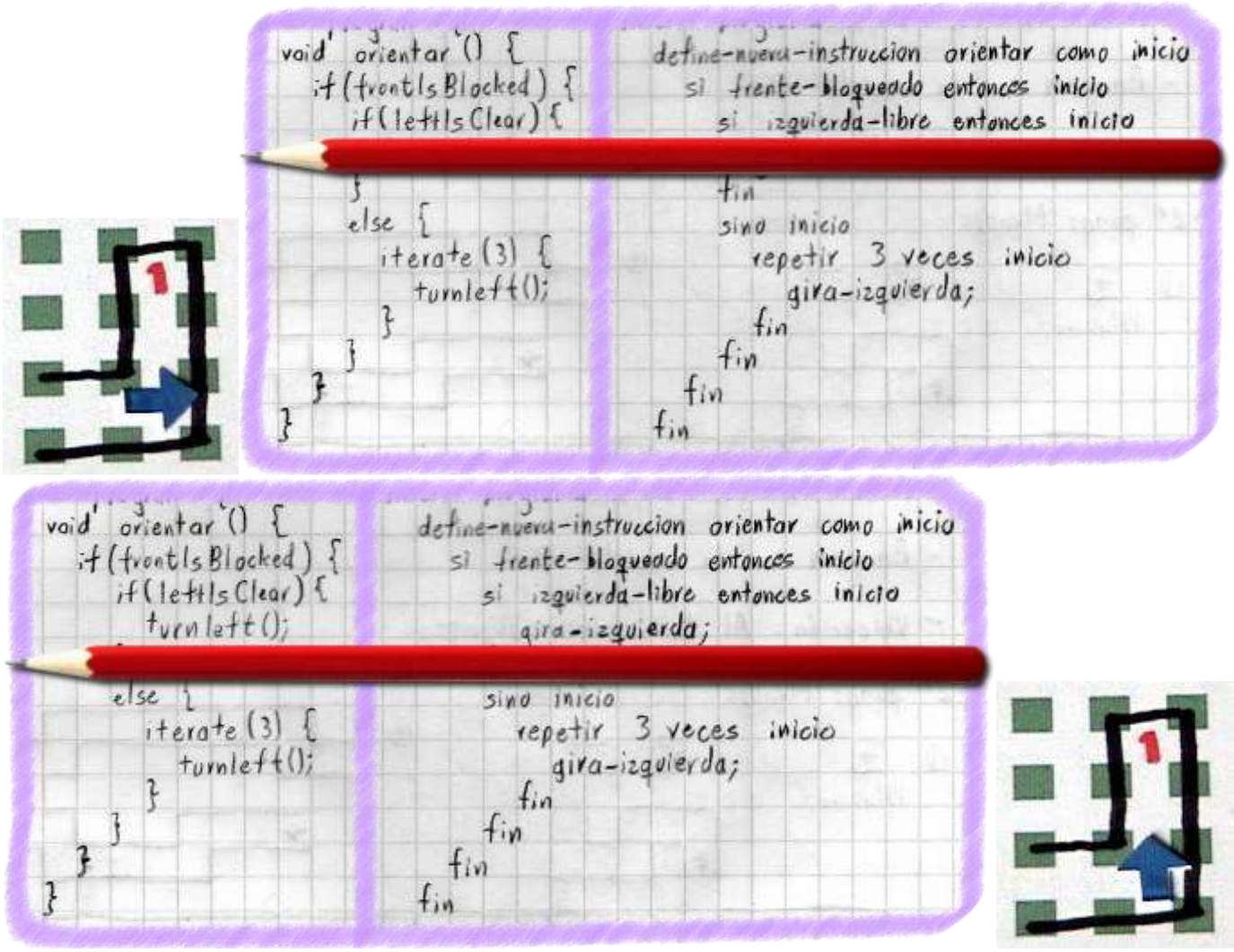
```
program () {
  while (not NextToABeeper) {
    orientar();
  }
  turnoff();
}
```

```
inicia-ejecucion
  mientras no-junto-a-zumbador hacer inicio
    orientar;
  fin;
  apagate;
termina-ejecucion
```



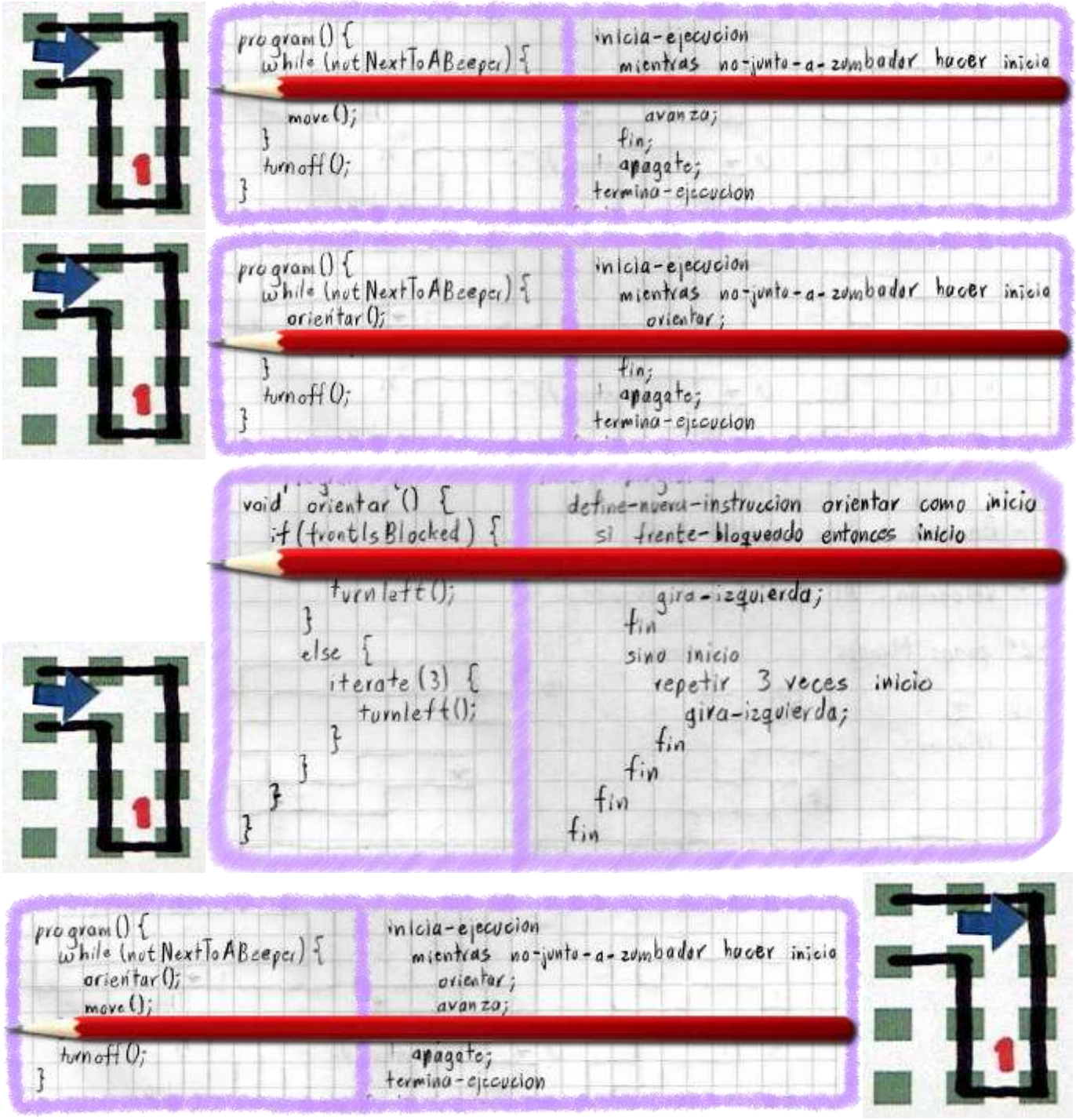
```
void orientar() {
  if (frontIsBlocked) {
    turnleft();
  }
  else {
    iterate (3) {
      turnleft();
    }
  }
}
```

```
define-nueva-instruccion orientar como inicio
  si frente-bloqueado entonces inicio
    gira-izquierda;
  fin
  sino inicio
    repetir 3 veces inicio
      gira-izquierda;
    fin
  fin
fin
```

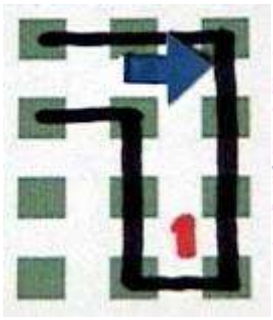


Dejándonos, finalmente, en la posición correcta para que siga su recorrido.

Ahora haremos la prueba que cumpla en el caso de que Karel se deba mover hacia la derecha. Como se muestra a continuación, nuestra rutina de control hace que Karel camine un paso.

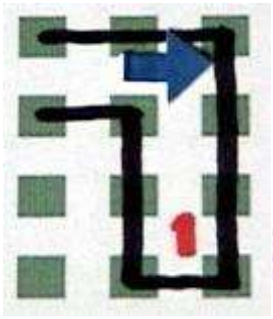


Dejándonos en esta posición, se llama a nuestra rutina *orientar*, la cual pregunta si el frente está bloqueado, esto siendo afirmativo; entonces pregunta si la izquierda está libre, siendo negativo para este caso; entonces, Karel hará 3 giros a la izquierda:



```
program() {
  while (not NextToABeeper) {
    orientar();
  }
  turnoff();
}
```

```
inicia-ejecucion
  mientras no-junto-a-zumbador hacer inicio
    orientar;
  fin;
  apagate;
termina-ejecucion
```

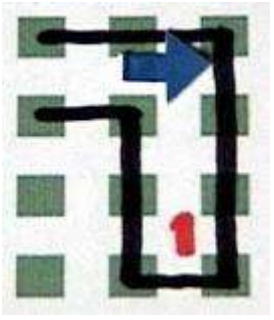


```
void orientar() {
  if (frontIsBlocked) {
    turnleft();
  }
  else {
    iterate(3) {
      turnleft();
    }
  }
}
```

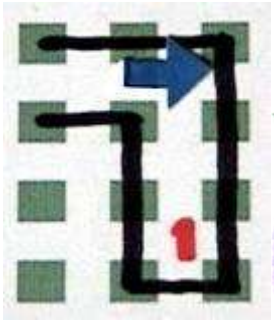
```
define-nueva-instruccion orientar como inicio
  si frente-bloqueado entonces inicio
    gira-izquierda;
  fin
  sino inicio
    repetir 3 veces inicio
      gira-izquierda;
    fin
  fin
fin
```

```
void orientar() {
  if (frontIsBlocked) {
    if (leftIsClear) {
    }
    else {
      iterate(3) {
        turnleft();
      }
    }
  }
}
```

```
define-nueva-instruccion orientar como inicio
  si frente-bloqueado entonces inicio
    si izquierda-libre entonces inicio
    fin
    sino inicio
      repetir 3 veces inicio
        gira-izquierda;
      fin
    fin
  fin
fin
```



Al final, nos dejará en la posición correcta.

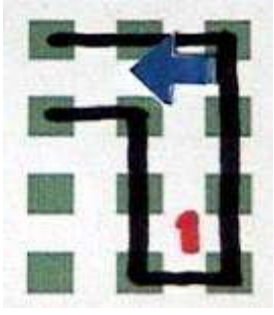


```
void orientar() {
  if (frontIsBlocked) {
    if (leftIsClear) {
      turnLeft();
    }
    else {
      iterate(3) {
      }
    }
  }
}
```

```
define-nueva-instruccion orientar como inicio
  si frente-bloqueado entonces inicio
    si izquierda-libre entonces inicio
      gira-izquierda;
    fin
    sino inicio
      repetir 3 veces inicio
      fin
    fin
  fin
fin
```

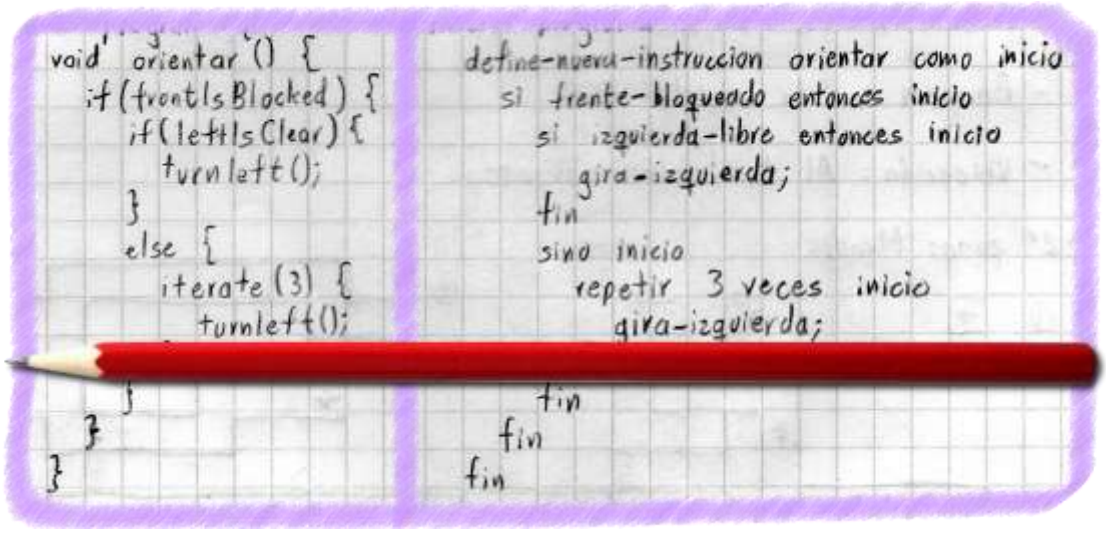
```
void orientar() {
  if (frontIsBlocked) {
    if (leftIsClear) {
      turnLeft();
    }
    else {
      iterate(3) {
        turnLeft();
      }
    }
  }
}
```

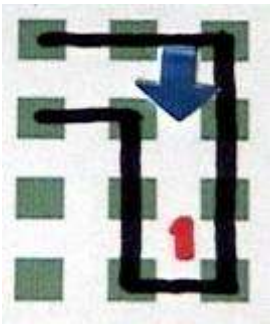
```
define-nueva-instruccion orientar como inicio
  si frente-bloqueado entonces inicio
    si izquierda-libre entonces inicio
      gira-izquierda;
    fin
    sino inicio
      repetir 3 veces inicio
        gira-izquierda;
      fin
    fin
  fin
fin
```



```
void orientar() {
  if (frontIsBlocked) {
    if (leftIsClear) {
      turnLeft();
    }
    else {
      iterate(3) {
        turnLeft();
      }
    }
  }
}
```

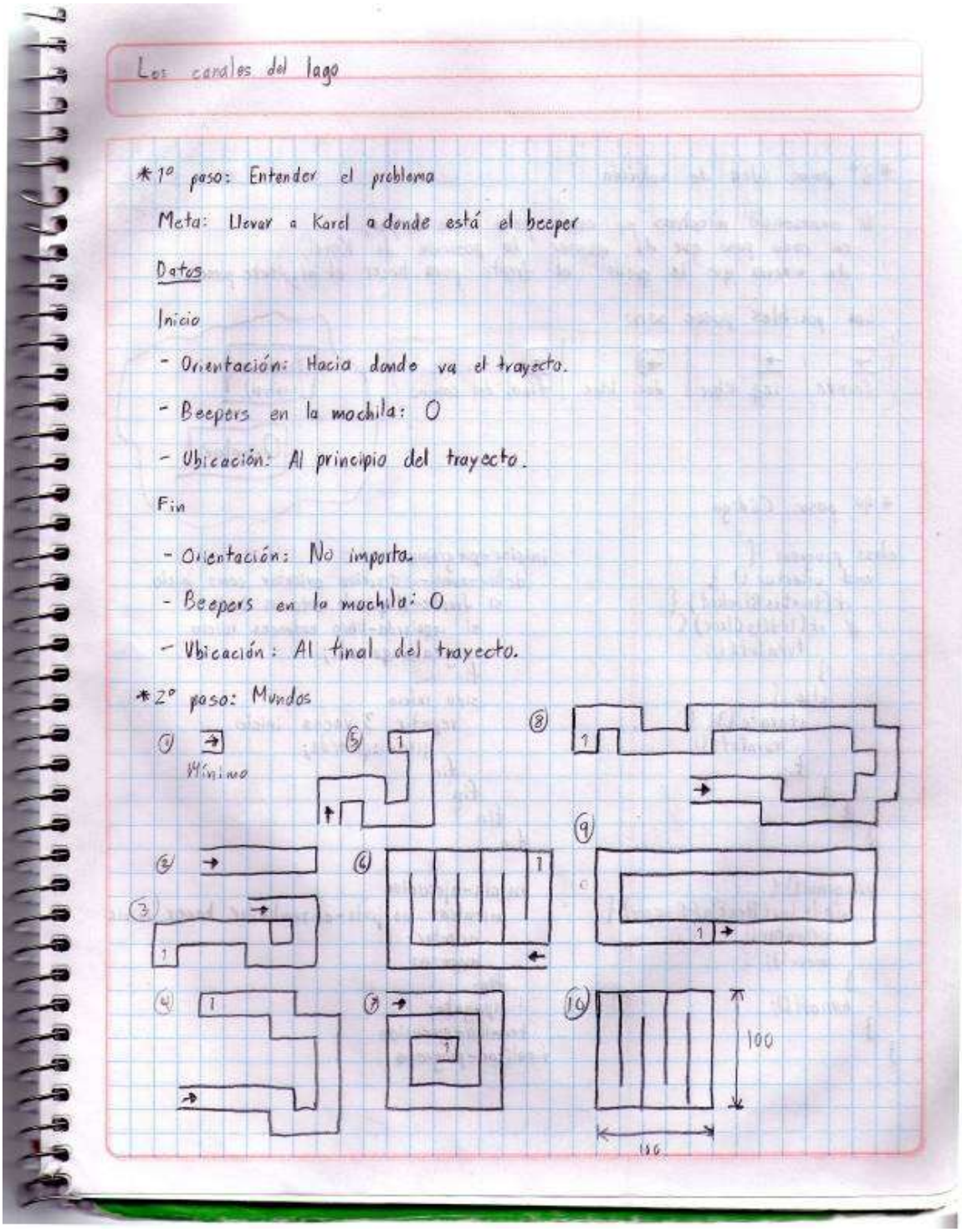
```
define-nueva-instruccion orientar como inicio
  si frente-bloqueado entonces inicio
    si izquierda-libre entonces inicio
      gira-izquierda;
    fin
    sino inicio
      repetir 3 veces inicio
        gira-izquierda;
      fin
    fin
  fin
fin
```





En este punto ya estamos seguros que nuestro programa va a funcionar bastante bien.

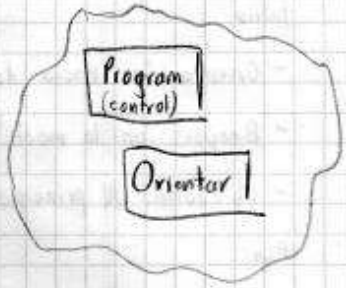
A continuación, te muestro las hojas de trabajo que se realizaron antes de hacer el programa en la computadora.



*3° paso: Idea de solución

Ir avanzando mientras no esté junto a un beeper;
en cada paso que dé, ajustar la posición de Karel,
de manera que le quede el frente para hacer el siguiente paso.

Los posibles pasos son:

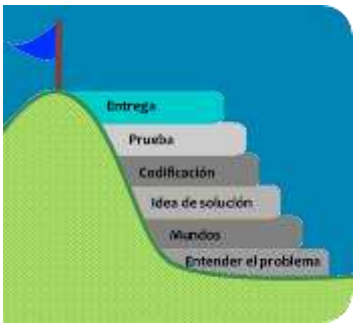


*4° paso: Código

```
class program {
  void orientar () {
    if (frontIsBlocked) {
      if (leftIsClear) {
        turnLeft();
      }
      else {
        iterate (3) {
          turnLeft();
        }
      }
    }
  }

  program() {
    while (not NextToABeeper) {
      orientar();
      move();
    }
    turnoff();
  }
}
```

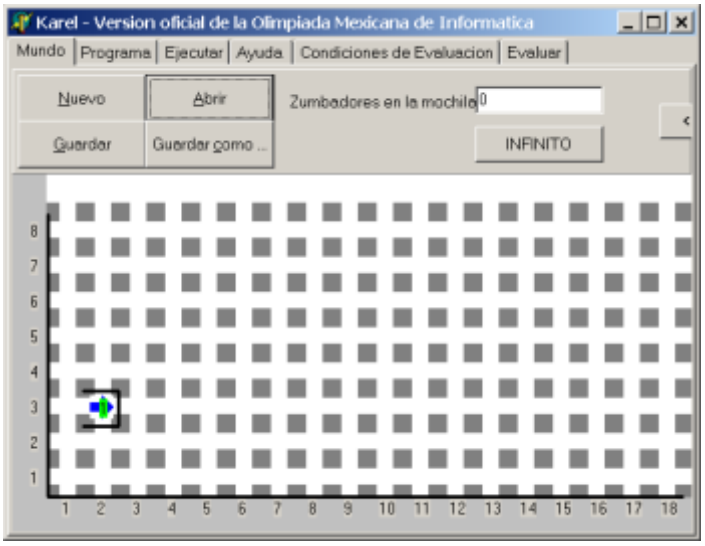
```
iniciar-programa
  define-nueva-instruccion orientar como inicio
    si frente-bloqueado entonces inicio
      si izquierda-libre entonces inicio
        gira-izquierda;
      fin
      sino inicio
        repetir 3 veces inicio
          gira-izquierda;
        fin
      fin
    fin
  fin

  inicia-ejecucion
    mientras no-junto-a-zumbador hacer inicio
      orientar;
      avanza;
    fin;
    apagate;
  termina-ejecucion
finalizar-programa
```

5.7. 6° Paso: Entrega

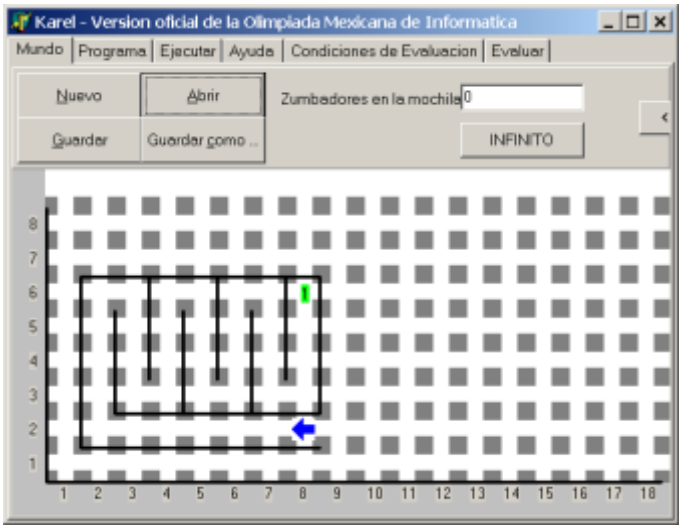


Lo que vamos hacer ahora es escribir los casos en el Karel. Recuerda que es importante guardar los mundos (casos), ya que debemos evitar estar reconstruyendo mundos si nuestro programa presenta una falla.

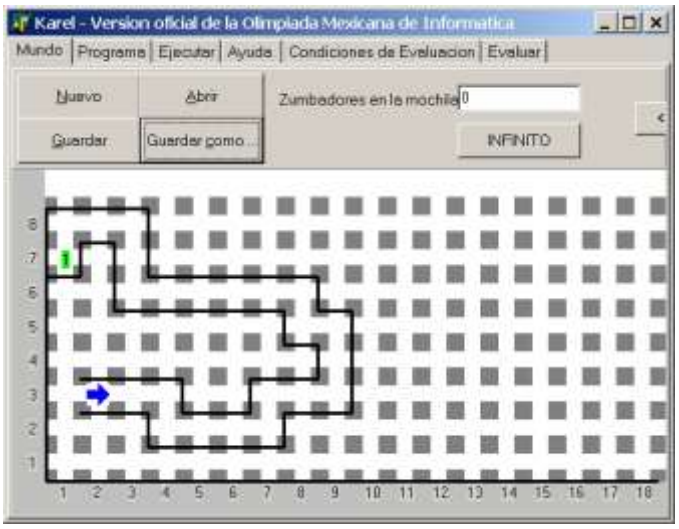
Una buena práctica es guardarlo con el nombre del problema seguido de un número.
A continuación se muestran las figuras de 3 mundos construidos (recuerda que debes hacer los 10 mundos y guardarlos):



Ejemplo del mundo mínimo, el cual tiene el nombre de archivo canales_01.mdo.

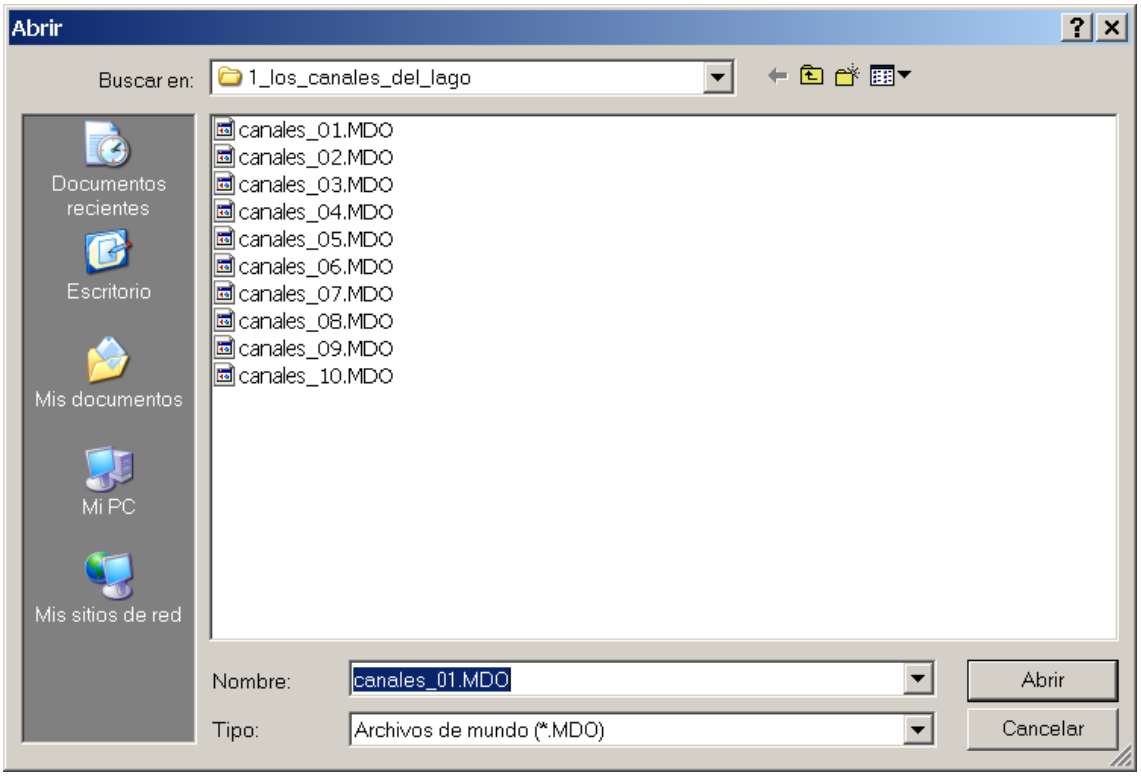


La siguiente imagen representa el caso promedio, representando el mundo número 6 (canales_06.mdo).

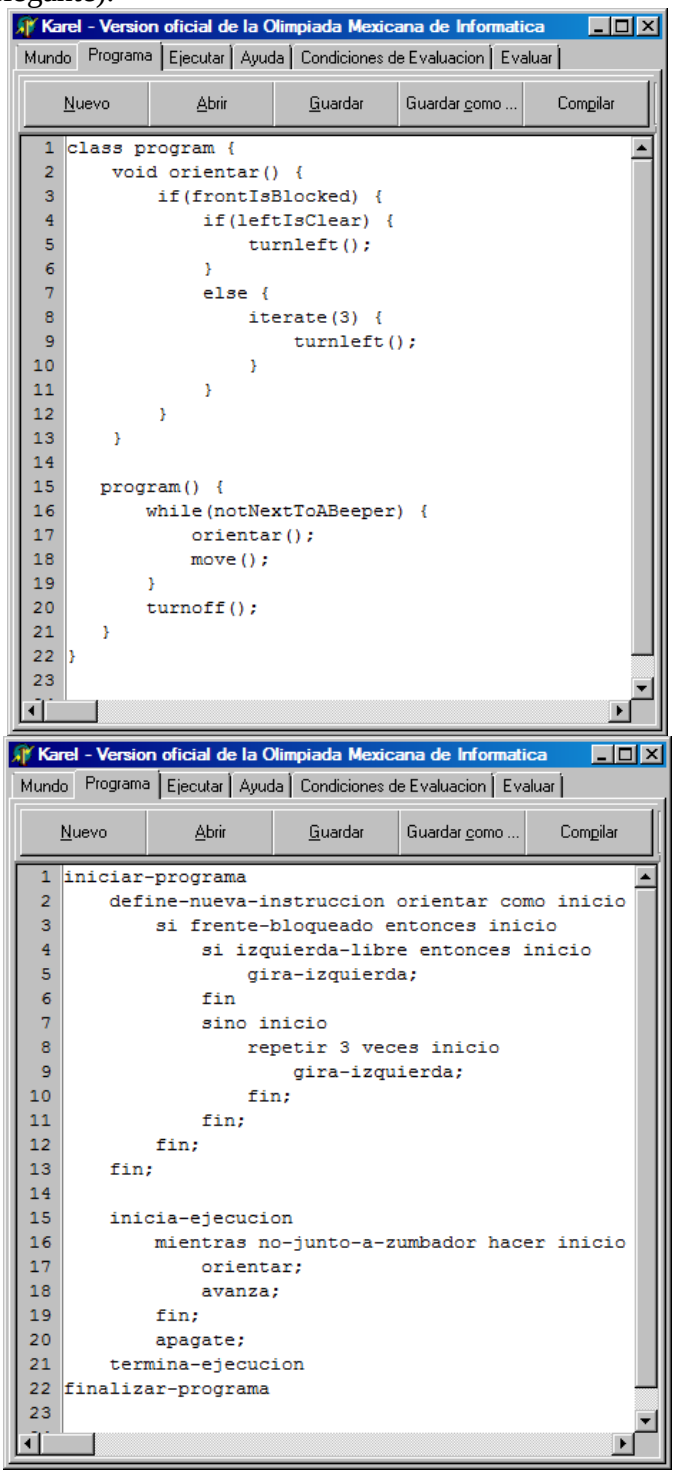


La siguiente figura representa el mundo 8 (canales_08.mdo). Este mundo representa todas las posibles variantes que puede tener en cuanto a cambios de orientación: hacia el norte, hacia el este, hacia el oeste y hacia el sur.

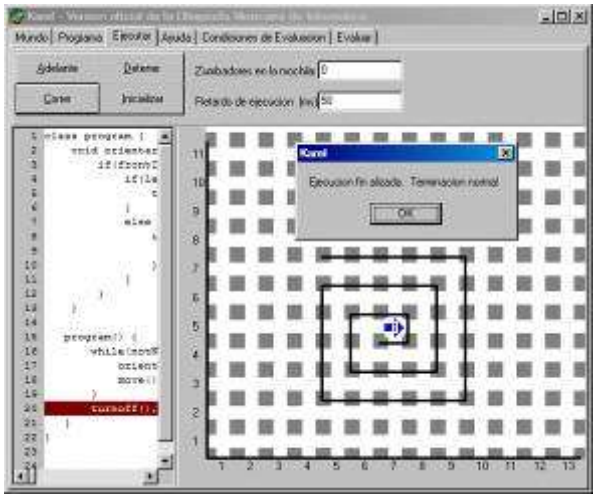
A continuación les muestro la imagen de abrir un mundo en donde vemos que ya están guardados los 10 mundos del problema "Los canales del lago".

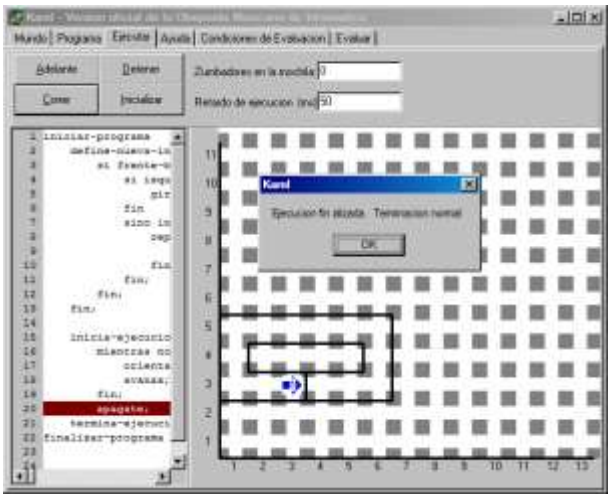


Posteriormente pasamos a escribir el programa en Karel, como se muestra en la siguiente figura. Debes fijarte que el programa se escribe indentando las instrucciones y como verás se ve muy claro (yo diría que hasta se ve elegante).



Ahora sólo falta probarlo con todos los mundos (casos) que fuiste capaz de pensar. Si al llegar a este punto descubres un error en tu idea de solución, te darás cuenta del tiempo que te ahorrarás al haber guardado tus mundos, ya que no los volverás hacer cada vez que tengas que probar tu programa. A continuación se muestran 2 mundos que fueron probados satisfactoriamente, tú sólo debes hacer el programa en un solo lenguaje; aquí muestro un ejemplo en Java y el otro en Pascal, porque este libro está dirigido a los profesores y alumnos de ambos lenguajes.





Por último, como podrás ver, tu programa funciona porque seguramente ya lo probaste con todos los casos y no tuviste error alguno, así que ya lo puedes entregar o continuar haciendo el siguiente problema, con la seguridad de que tu solución te dará los 100 puntos del problema. Te recomiendo que vayas midiendo el tiempo de elaboración de un programa; en cuanto domines el procedimiento para hacer un programa verás que es muy sencillo y rápido hacer un problema.

5.8. Tips para alumnos

A continuación te hago una lista en importancia de los tips que nunca debes de brincarte y que te ayudarán a lograr un mejor desempeño.

1. Construir tus casos antes de pensar en una solución; de esta manera tu solución te dará el 100% de los casos.
2. Por cada nueva instrucción que escribas debes hacer la prueba de escritorio, este punto te ayudará en un 30% de tu calificación y te ahorrará mucho tiempo.
3. Escribe tus mundos en Karel y guárdalos en caso de que tu programa falle. Así, no tendrás que reescribir los mundos para volverlo a probar, esto nos ahorrará tiempo y nos dará seguridad.
4. Entregar tu programa hasta que lo hayas probado con todos las variantes de mundos que fuiste capaz de encontrar.
5. Si no tienes experiencia en Karel, escribe todas las nuevas instrucciones en papel.
6. Trata de hacer tu Karel de papel para ayudarte en tu prueba de escritorio.
7. Indentar de manera adecuada tu programa para que sea más fácil de entenderlo en caso de que debas ajustar algunas instrucciones.

5.9. Tips para profesores

A continuación te hago énfasis en lo que ayudará a mejorar el desempeño de tus alumnos en las clases.

1. Primero, debes creer en lo que estás haciendo. En caso de no estar totalmente de acuerdo con esta guía, te invito a que la modifiques de acuerdo a tu experiencia y grado escolar, y me dará mucho gusto que la compartas conmigo.
2. Ser muy estricto en cuanto al procedimiento a seguir. Te sugiero que les apagues las pantallas de sus computadoras hasta que te presenten en su cuaderno los datos del problema y los casos y su solución. Una vez que hagan esto, tú mismo les enciendes el monitor.
3. Ser exigente con la identación para que los chicos se hagan de un buen hábito al escribir programas de computadora.
4. Si los chicos no tienen mucha experiencia o son muy pequeños, te sugiero que les entregues en hojas de papel blanco el mundo de Karel y los hagas construir un Karel de papel a fin de poder realizar las pruebas de escritorio de sus programas de una manera sencilla y amena. En el anexo 1 del libro existe una hoja de papel con el la imagen del mundo de Karel y un Karel listo para fotocopiar.
5. Te recomiendo mucho leer los problemas que vas a poner y revisar las soluciones antes de presentarte ante el grupo.

6. Parámetros en funciones

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** pusiésemos un número variable, podríamos usar la instrucción tanto para girar a la derecha como para dar media vuelta.

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 qué 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 instrucción 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.

Aquí 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.

Guía de Ejercicios

Ejercicio 1: Ayudando a mamá

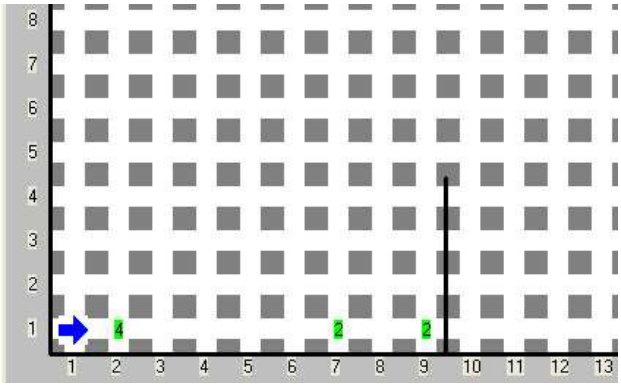
Karel ha crecido un poco y ahora su mamá le ha pedido que recoja la basura (zumbadores) de su casa y se los tire al vecino.

Consideraciones

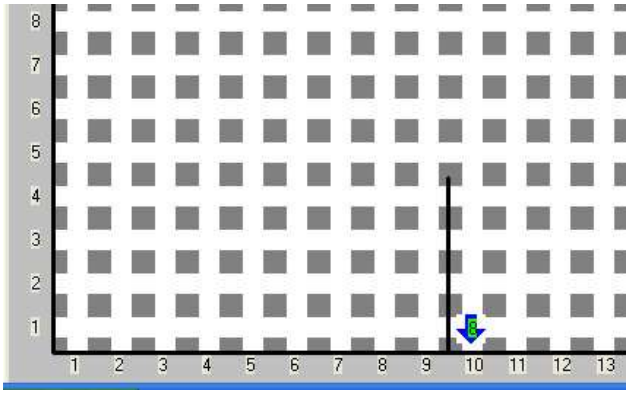
- Karel puede empezar en cualquier posicion de su casa
- No sabemos cuanta basura habra en cada lugar
- Karel sabe que termino de recogerlos al llegar a la barda de su casa
- La barda tiene una altura indeterminada.
- Karel debera dejar la basura en el piso de la casa del vecino
- No importa en donde ni hacia donde queda apuntando karel

Ejemplo

Mundo inicial



Mundo final



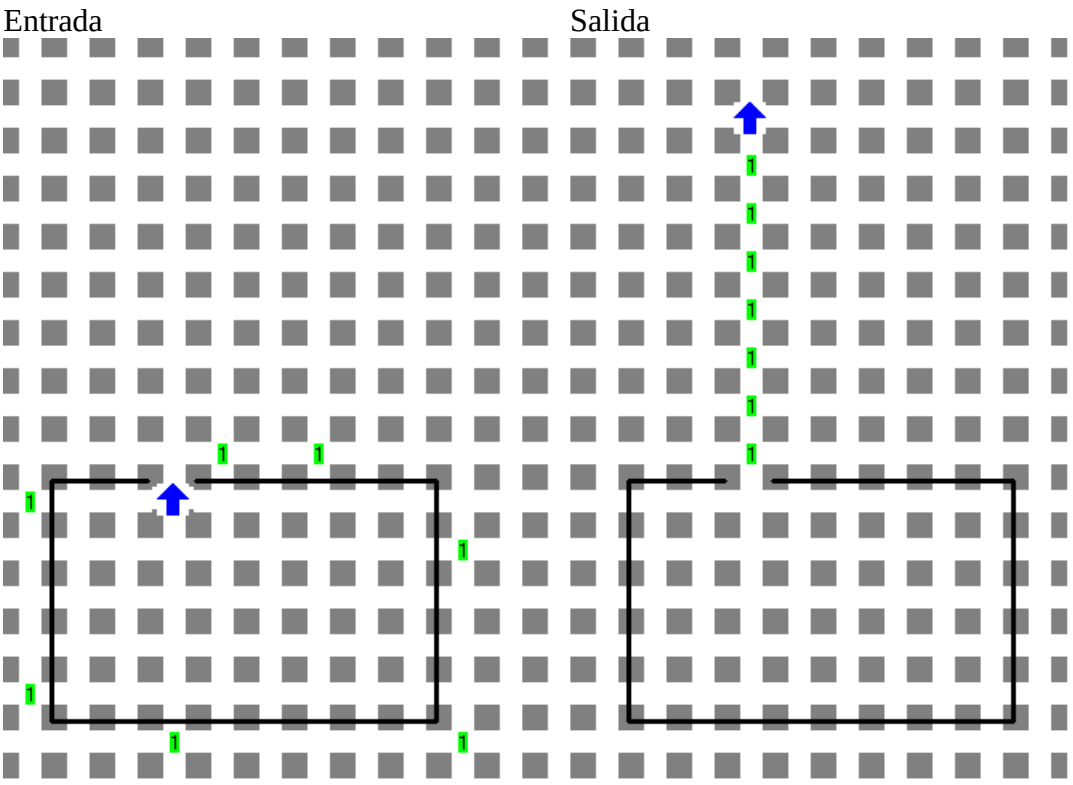
Ejercicio 2: Rekreo

Descripción
Después de 2 horas de estudio el maestro les ha dado a los alumnos tiempo de recreo, el cual acaba de terminar, la campana del Colegio no sirve por lo que requerimos tu ayuda.

Problema
Deberás salir al patio y decirle a los alumnos que ha terminado el recreo y deben regresar al aula para la segunda parte de la clase. El sol esta fuerte así que todos los alumnos se encuentran recargados en la pared del salón que da hacia el patio de juego.
Los alumnos deberán hacer una fila en la puerta de entrada del salón

Consideraciones

- Tu programa será evaluado con distintos casos de prueba
- Karel inicia en la puerta del salón orientado hacia la salida
- Los estudiantes están representados con 1 zumbador
- Todos los alumnos se encuentran en la parte externa del salón pegados a la pared
- El salón es rectangular o cuadrado
- Desconoces las dimensiones del salón
- Karel deberá terminar posicionado después del último alumno de la fila, sin importar su orientación



Ejercicio 3: ¡Tache!

Descripción
Karel vive dentro de un cuadrado.

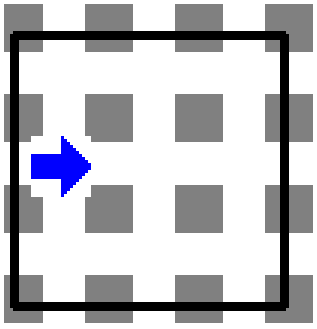
Problema
Tu tarea consiste en poner un tache dentro del cuadrado.

Consideraciones

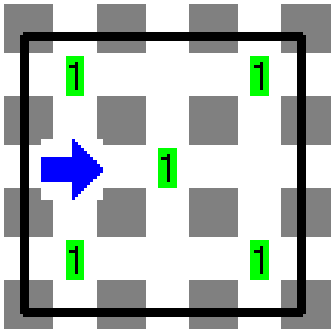
- Karel tiene la cantidad suficiente de zumbadores para hacer su tarea en la mochila.
- No habrá paredes dentro del cuadrado.
- No habrá zumbadores dentro del cuadrado.
- La posición y orientación inicial de Karel es desconocida.
- Al terminar no importa la orientación ni posición de Karel.

Ejemplo:

Entrada



Salida



Ejercicio 4: Medias pirámides

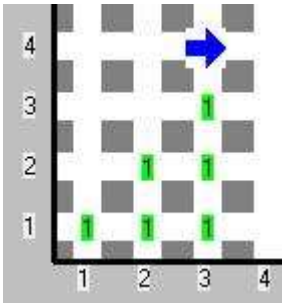
Descripción
Es bien sabido que en la ciudad e Karelotitlán todos los habitantes tienen sus casas en forma de pirámides, sin embargo, el excéntrico Karelerac quiere ser diferente a los demás y desea construir su casa con sólo la mitad de una pirámide.

Problema
Ayuda a Karelerac a construir una media pirámide creciente hacia la derecha con los zumbadores que lleva en la mochila.

Consideraciones

- Karel empieza en la posición (1,1) orientado al norte.
- Karel lleva en la mochila un número "triangular" de zumbadores, por lo que nunca se quedará sin zumbadores durante la construcción.
- Karelerac es un nombre palíndromo (o casi palíndromo).
- No importa la posición ni la orientación final de Karel.

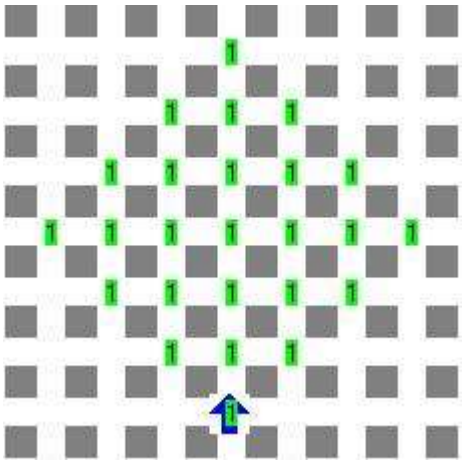
Ejemplo:



Ejercicio 5: Cosecha de Baseball

Nuestro querido Karel sembro un campo de beepers despues de ir a un partido de Baseball. El trabajo de Karel será hacer una cosecha de un campo en forma de rombo.

- Karel inicia siempre en la parte inferior del rombo mirando al Norte.
- El tamaño del campo no esta definido.
- El trabajo termina cuando ha recogido todos los beepers
- Pueden existir muros en la frontera del rombo de beepers



Cosechando el campo de Baseball

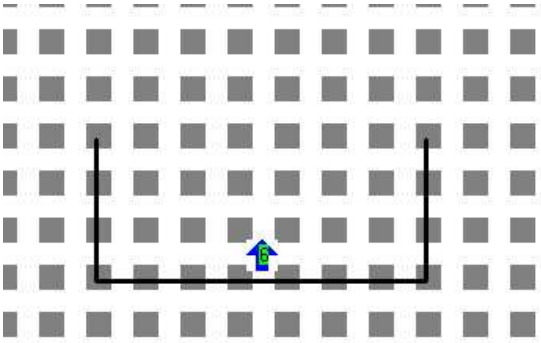
Ejercicio 6: “Repartiendo juguetes”

Karel esta jugando con su amigo mando, tienen juguetes y los van a repartir, ayudale a karel

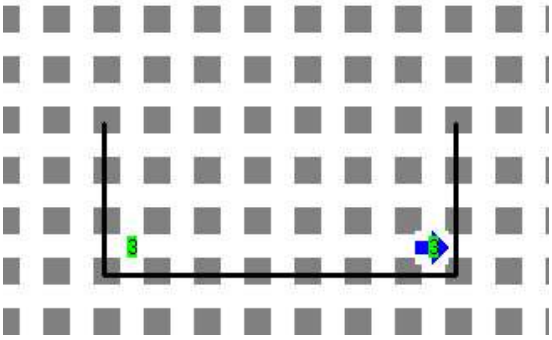
- Karel inicia apuntando hacia el norte sobre un monton de juguetes
- Las dimensiones del cuarto puede ir de 3 a 20.
- Karel debera recogerlos y repartirlos, los juguetes de karel en la esquina izquierda y los de mando en la esquina derecha
- Si existe un numero impar de juguetes, karel decide colocar el ultimo juguete en su esquina.
- Karel debe acabar en su monton de juguetes, pero no importa hacia donde queda apuntando

Ejemplo:

Mundo inicial



Mundo final



Ejercicio 7: EduKarel

Descripción

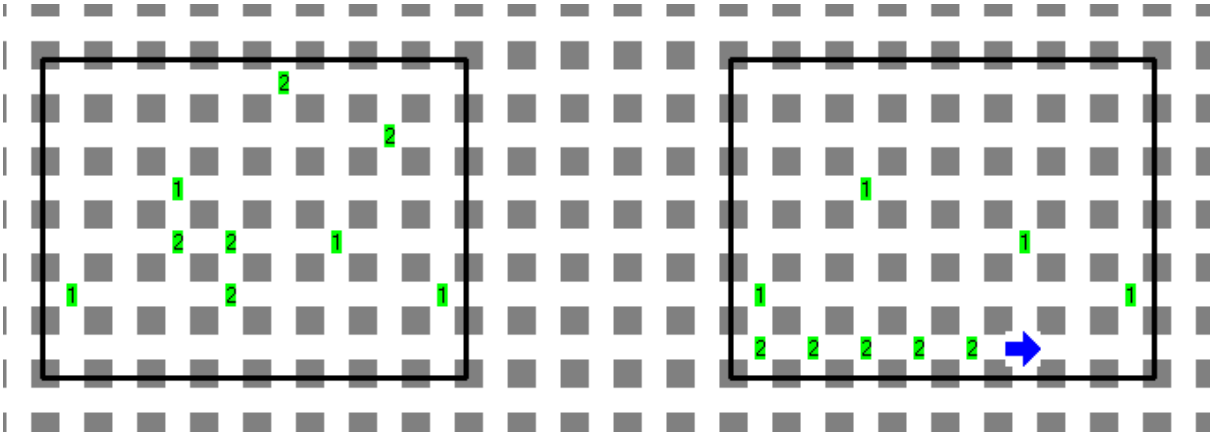
En OMiJal nos hemos dado cuenta de la falta de material de estudio para aprender KAREL, por ellos estamos preparando una serie de actividades que nos llevara a crear un sitio WEB exclusivo de Karel, con tutoriales, ejercicios, problemarios, etc. Para ello, tu nos ayudaras en la planeación
Actualmente cada quien aprende Karel como puede, tu misión es buscar a los interesados en aprender Karel y llevarlos a un aula donde se les dara un taller introductorio.

Problema

Ayuda a EduKarel a localizar a los interesados en aprender (2 zumbadores) y llévalos al aula!

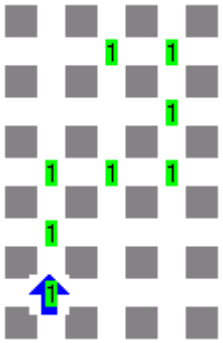
Consideraciones

- Tu programa será evaluado con diversos casos de prueba
- El mundo está delimitado por paredes y es de forma rectangular/cuadrada
- Karel inicia en la parte inferior izquierda, viendo hacia el norte
- Los alumnos interesados en aprender Karel tienen como característica ser el doble de inteligentes (2 zumbadores) que el resto de la población (1 zumbador)
- El "aula" es el primer renglón donde necesariamente siempre cabrán los alumnos interesados
- Karel deberá dejar a los zumbadores en la primer línea, uno seguido a otro
- El primer renglón (el aula) siempre inicia con alumnos (zumbadores)



Ejercicio 8: Siguiendo una llínea.

Para este problema el mundo de Karel constará de una línea formada por cuadros consecutivos con 1 zumbador, como el que se muestra en la figura.



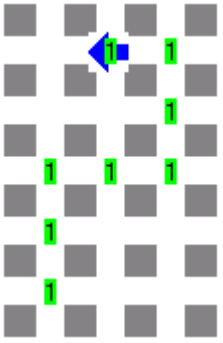
La línea, como se muestra, puede tener vueltas. Las vueltas, en caso de que las haya, siempre serán de 90 grados. La línea nunca se cruza sobre si misma, y siempre tiene un ancho de una posición. En el caso en que una línea diera vuelta en "U" siempre habrá al menos una casilla en blanco separando la ida y la vuelta. La línea siempre tendrá un inicio y un final.

Independientemente de donde este situada la línea, Karel siempre estará posicionado en uno de los extremos de la misma.

Deberás escribir un programa que haga que Karel recorra la línea y se sitúe en el otro extremo de la misma.

Tu programa obtendrá puntos si Karel termina en la posición donde esta el otro extremo de la línea sin importar hacia donde este orientado.

Por ejemplo, para la figura anterior, tu programa obtendrá puntos, si al final de la ejecución Karel esta en la posición que se muestra en la figura 2. **No importa la orientación, sólo la posición.**



La línea puede estar en cualquier posición dentro del mundo, y nunca tendrá un largo mayor a 500 casillas.

Ejercicio 9: Igualando las columnas

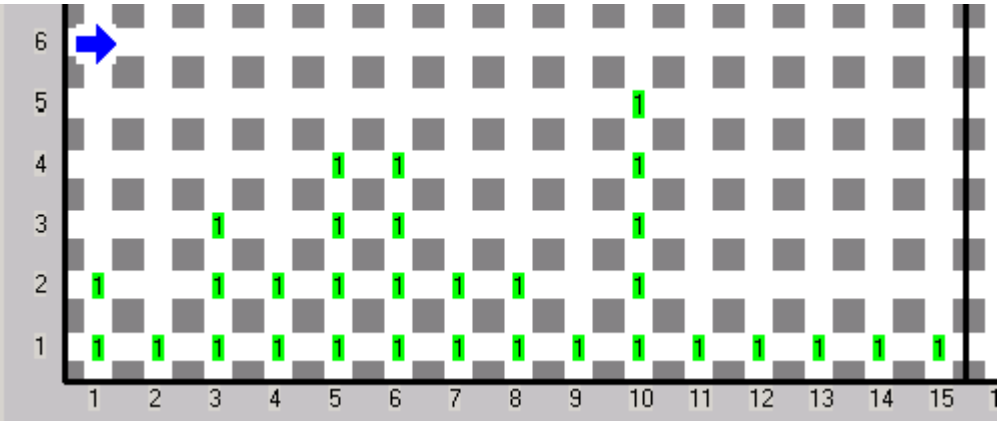
Descripción:

Incialmente Karel se encuentra orientado al ESTE en la posición (1,6) de su mundo. Comenzando por la primera columna donde esta Karel y hacia la derecha hay 10 columnas de zumbadores de diferentes alturas. Cada columna es una secuencia ininterrumpida de posiciones con 1 zumbador (ver figura de ejemplo). Las columnas tienen alturas diferentes que pueden ir desde 1 hasta 20.

A la derecha de las 15 columnas hay una pared vertical que se eleva hasta la posición 30.

Debes escribir un programa que haga que Karel iguale la altura de todas las columnas pasando zumbadores de una columna a otra.

Importante: No debe de haber mas de un zumbador por posición.
NOTA: En los casos de prueba que se utilizarán para calificar tu programa, el número de zumbadores será tal, que siempre se podrá igualar la altura de todas las columnas.
Ejemplo:



Ejercicio 10: Karel el conquistador

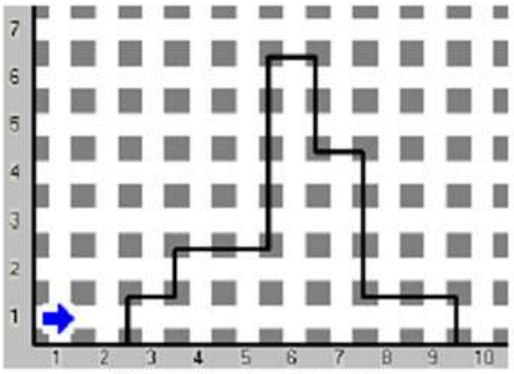
Karel se encuentra frente a una montaña, debe conquistar el punto más alto y dejar una bandera. La bandera estará representada por un zumbador.
Especificaciones:

Entrada

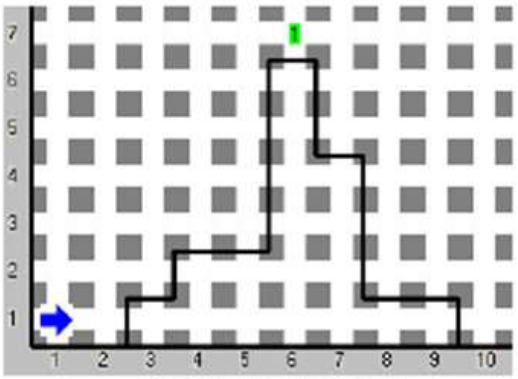
Mundo: Hay una sola montaña, que estará representada por paredes de forma escalonada cuyos escalones pueden ser de diversas alturas y longitudes. La montaña tiene solamente un pico.
Mochila: Tiene un único zumbador.
Posición: (1,1)
Orientación: Este.

- **Salida**

Mundo: El mundo inicial con un zumbador en el pico.
Mochila: Vacía.
Posición: No importa.
Orientación: No importa.



Ejemplo de entrada



Ejemplo de salida

Tarea 1: Karel Traviesa

Historia

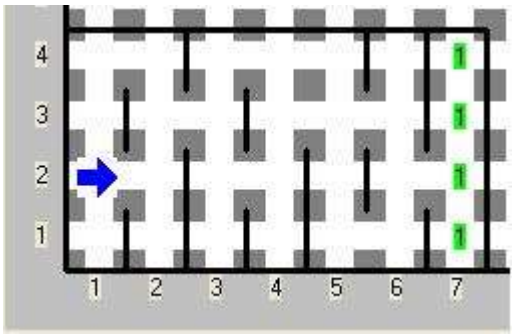
El ciclista Karel Armstrong se inscribió a una carrera a Mundo-traviesa. Esta carrera la gana el que llegue primero desde la primera avenida hasta la última avenida. La meta está formada por toda una fila vertical de beepers (en la última avenida). En el camino desde la primera avenida hasta la meta hay obstáculos (paredes verticales) que impiden el paso al ciclista Armstrong. Sin embargo, siempre existe una forma de seguir avanzando (es decir, nunca hay una pared corrida desde la primera hasta la última calle).

Problema

Escribe un programa que lleva a Karel desde su posición inicial (siempre volteando hacia el este) hasta la avenida llena de beepers.

Consideraciones

- Karel inicia siempre en la primera avenida.
- El número de calles estan limitado por paredes.
- La carrera termina cuando llega a una avenida con beepers.
- El número de avenidas es igual o mayor a dos.



Tarea 2: DISTRIBUYE ZUMBADORES

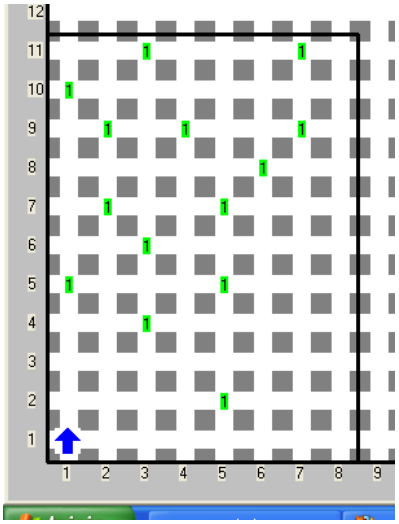
DESCRIPCION

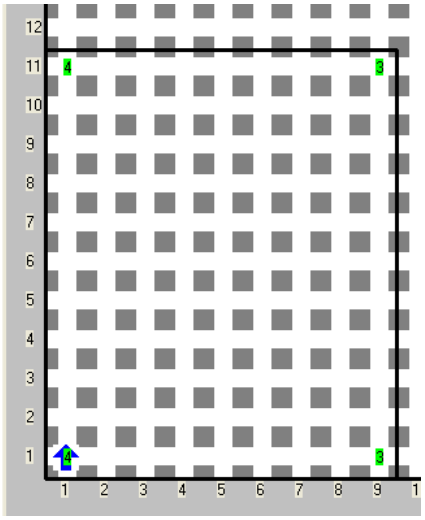
Karel tuvo una gran fiesta en su casa el fin de semana, y en varias partes de su casa quedaron restos de basura, Karel esta preocupado que sus padres lo descubran, por lo que decidió recoger toda la basura y tirarla en diferentes botes de basura de una manera equivalente.

PROBLEMA

Tu tarea consistirá en construir un programa que ayude a Karel a recoger la basura, que se representa con “n” montones de zumbadores con “x” zumbadores cada uno, y distribuirlos equitativamente en los 4 botes de su casa que se encuentran en las esquinas empezando por el bote de la esquina inferior izquierda y siguiendo por el bote del norte.
Inicialmente, Karel se encuentra en la posición (1,1) orientado hacia el norte. La casa de Karel puede ser cuadrada o rectangular con un tamaño desde (1,1) hasta (99,99)

Inicialmente no puede haber basura tirada (zumbadores), en los botes, no importa la orientación ni posición final de Karel.



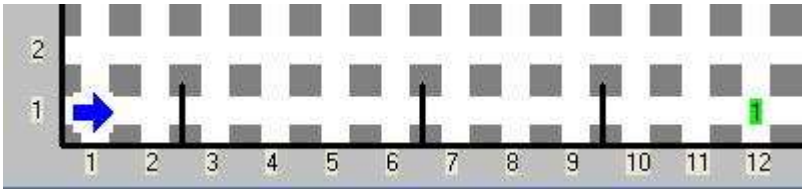


Tarea 3: Diez Avenidas con Vallas

Karel se encuentra en una competencia de carrera. Es necesario que recorra un camino que tiene vallas (obstáculos) hasta llegar al final donde se encuentra un beeper. El problema tiene las siguientes ordenes:

- Karel inicia siempre al principio de la pista mirando al Este (dirección a la que corre) y pegado a la derecha.
- Karel siempre debe ir lo más apegado que pueda a la pared derecha.
- Las vallas siempre están sobre las avenidas (verticales), son de longitud uno y no hay dos que estén en avenidas consecutivas.
- El número de vallas no esta determinado (no puedes saber cual es la configuración de las vallas).
- La longitud de la pista no esta determinada.
- La carrera termina cuando llega a un beeper.

EJEMPLO 1



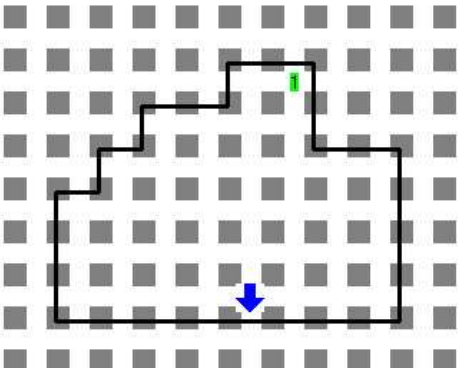
Tarea 4: "LA HUIDA"

A KAREL SE LE PASO LA MANO Y MANDO HA PASADO A MEJOR VIDA, POR LO TANTO KAREL TIENE QUE HUIR, Y SU NOVIA KAROLA ESTA ESPERANDOLO EN EL PUNTO MAS ALEJADO DE NUESTRO PAIS, TU MISION ES AYUDAR A KAREL A ENCONTRARSE CON KAROLA (1)

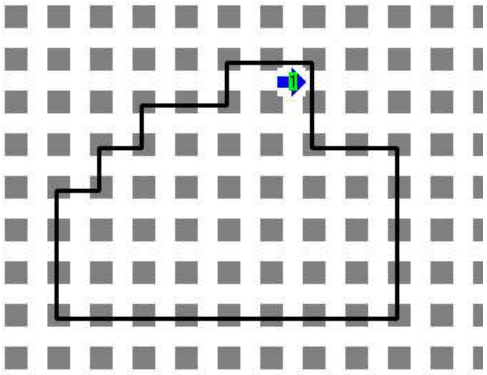
- EL MUNDO PUEDE TENER CUALQUIER FORMA EN LA PARED NORTE
- KAREL INICIA APUNTANDO HACIA EL SUR EN EL CENTRO DEL MUNDO
- KAREL DEBERA IR A DONDE ESTA KAROLA(1)
- NO IMPORTA HACIA DONDE QUEDA APUNTANDO KAREL

EJEMPLO:

MUNDO INICIAL



MUNDO FINAL



Tarea 5: Pintando la ciudad

Historia

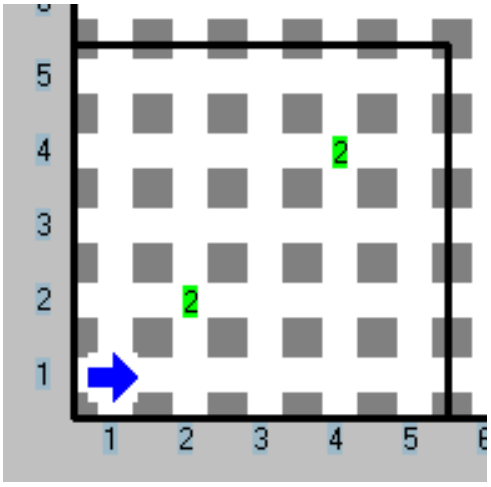
En la ciudad natal de Karel, siempre se han preocupado por la estética de las calles, por lo que pintan cada dos meses algunas de ellas, no siempre son las mismas, sólo las más deterioradas, y Don Karlopolus (el papá de Karel) es el encargado de hacerlo. En esta ocasión, por problemas de salud ha pedido a Karel pintarlas, y solo le ha dejado una marca que señala la intersección de las calles que tendrá que pintar por completo.

Problema

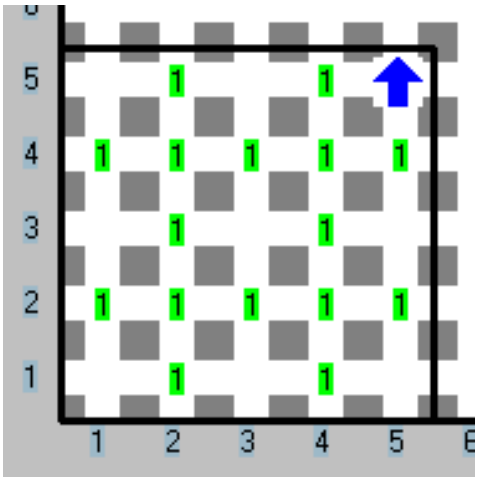
Karel tiene que pintar las calles y tienes que ayudarle. Una calle pintada esta llena de zumbadores (una línea de zumbadores en 1), y las marcas que dejó el papa de Karel son dos zumbadores. La ciudad siempre es de forma rectangular y está delimitada por paredes.

Consideraciones:

1. Karel inicia en la esquina inferior izquierda.
2. Karel tiene que dejar toda la calle pintada, no puede dejar las marcas que su papa puso.
3. Los zumbadores que están en la ciudad son siempre de tamaño uno o dos.
4. Karel lleva infinito número de zumbadores.
5. No sabes cuantos habitantes tiene la ciudad.
6. No existen paredes dentro de la ciudad.
7. No importa la posición ni orientación final de Karel



Estado inicial de ejemplo



Solución

Tarea 6: ZUMBADOR POR ZUMBADOR

Descripción

Karelandia ha tenido unas elecciones para reina de la primavera, pero han quedado dudas sobre el conteo, ambas candidatas a reina de Karenlandia (Diana y Elba) han pedido una revisión ya que sospechan que hubo votos "perdidos"

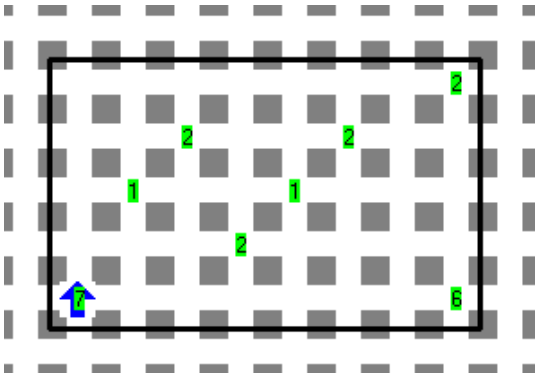
Problema

Ayuda a KarelIFE a localizar votos (zumbadores) perdidos y dárselos a la candidata correcta.
Los votos 1 son de Diana (esquina inferior izquierda), los Votos 2 son de Elba (esquina inferior derecha)

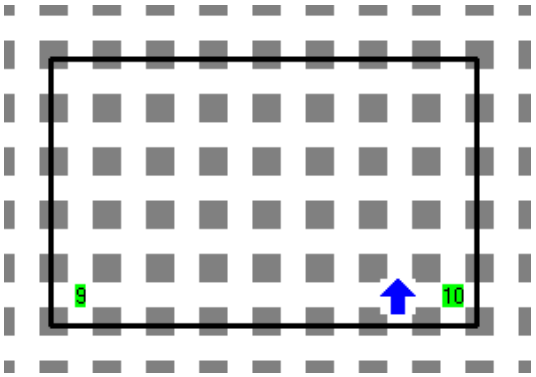
Consideraciones

- Tu programa será evaluado con 5 distintos casos de prueba
- Karel inicia en la parte inferior izquierda, donde están los votos de Diana
- Los votos perdidos (1) son de Diana, los (2) de Elba
- No hay votos perdidos en la primera línea
- El mundo está delimitado por paredes.
- Los votos (1 o 2) valen lo mismo por lo que si encuentras votos de Elba (2) solo deberás agregarle 1 a su votación.

Ejemplo:



Entrada

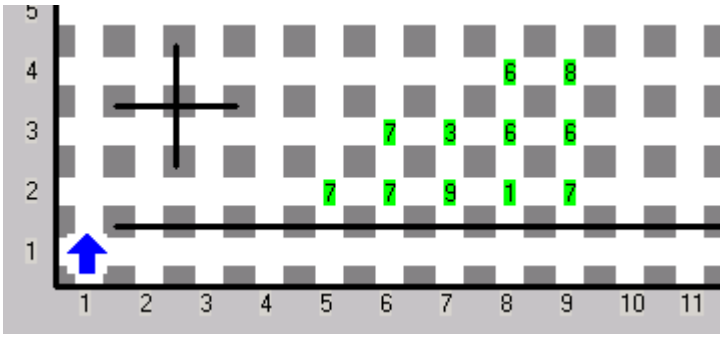


Salida

Tarea 7: Sumar.

Para este problema deberás hacer un programa que sume un conjunto de números, que puede tener desde 2 hasta 10 elementos.

El mundo inicial, tiene la configuración que se muestra en la siguiente



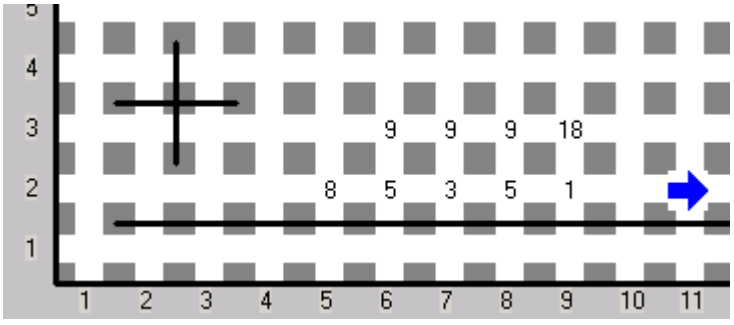
Karel esta posicionado inicialmente en la posición (1,1) orientado hacia el norte. La línea horizontal que se muestra entre las filas 1 y 2 se extiende hasta el final del mundo.

Los números que se tienen que sumar se encuentran en las filas 2, 3, ... Todos los números tienen su cifra menos significativa en la columna 9, y cada número nunca excederá a las 5 cifras. Los números siempre estarán ordenados por número de cifras, con el número con más cifras hasta abajo.

Ningún número tendrá 0 como una de sus cifras.

Tu programa deberá hacer que Karel sume los números y escribir el resultado de la suma en la fila 2, con el dígito menos significativo en la columna 9.

Por ejemplo para el mundo que se muestra en la figura anterior el resultado correcto sería



No importa en donde quede Karel, ni su orientación ni los zumbadores que existan en otras posiciones del mundo, siempre y cuando en la fila 2, a partir de la columna 9 se encuentre la suma de los números.

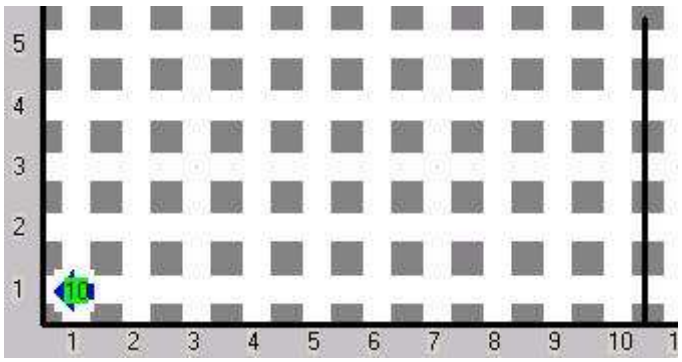
Ejercicio recursividad 1: Midiendo la distancia

Problema
Debes escribir un programa que permita que Karel mida la distancia entre su posición inicial y una pared. Como resultado, Karel deberá dejar en la esquina inferior izquierda del mundo un beeper por cada calle que este entre ésta y la pared.

Consideraciones

- Inicialmente Karel se encuentra orientado hacia el ESTE en la esquina inferior izquierda de su mundo.
- A la derecha de Karel, a una distancia desconocida hay una pared vertical.
- Karel Tiene 100 beepers en su Beeper Bag.
- No importa la dirección y posición con la que termine Karel.

Ejemplo:



Ejercicio Recursividad 2: Leyendo el periódico

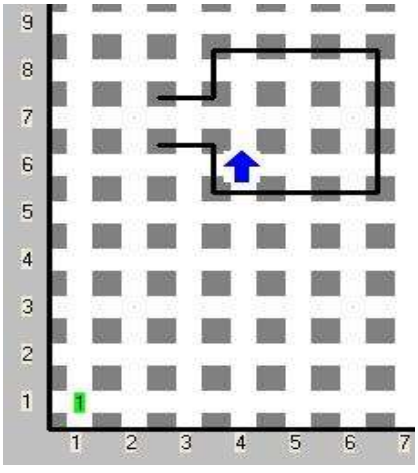
Descripción
Después de una larga noche de sueño, Karel se ha levantado y se ha dirigido a su sala dispuesto a leer el periódico.
Sin embargo, este día el repartidor no dejó el periódico en la puerta de su casa, esta vez, lo dejó en la esquina de la calle porque Karel no le ha dado propina.

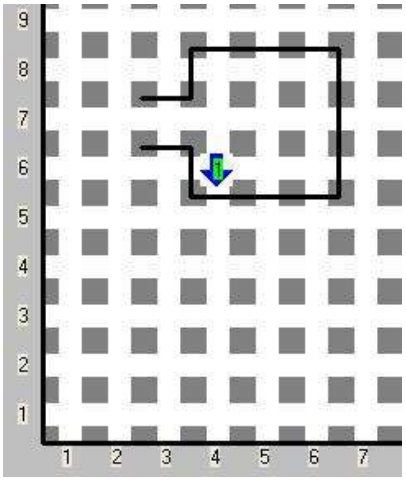
Problema
Escribe un programa que permita a Karel ir por el periódico y regresar a su sala a leerlo tranquilamente.

Consideraciones

- La casa de Karel se encuentra en algún lugar del mundo, está representado por un rectángulo de paredes de 3 X 3 y tiene una salida en la pared izquierda como se muestra en el mundo de ejemplo.
- La sala de la casa de Karel, se encuentra en la esquina inferior izquierda de su casa.
- Karel está inicialmente en la sala de su casa.
- El periódico se encuentra en la esquina (1,1) del mundo representado por un beeper.
- Karel no tiene zumbadores en la mochila.
- Karel está muy interesado en leer los anuncios clasificados.
- Karel debe de tomar el periódico y regresar a su sala, en donde debe poner el periódico (beeper) en el suelo.
- No importa la orientación final de Karel.
- Además de la casa de Karel y del periódico, no hay más paredes o beepers en el mundo.

Entrada **Salida**





Ejercicio Recursividad 3: Karel Topografo

Karel debe calcular el perímetro de un terreno rectangular.
Especificaciones:
Entrada

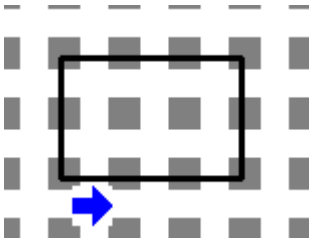
Mundo: El rectángulo puede estar situado en cualquier parte del mundo.
Los lados del rectángulo son verticales y horizontales. El lado izquierdo estará después de la avenida 1 vertical, aunque no necesariamente en la avenida 2. El lado inferior estará después de la avenida 1 horizontal. Aunque no necesariamente en la avenida 2.

Mochila: Infinito zumbadores.
Posición: Karel se encontrara exactamente en la parte inferior izquierda del rectángulo.
Orientación: Este.

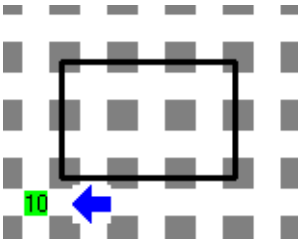
Salida

Mundo: En la casilla izquierda de la que inicio Karel tantos zumbadores como unidades sea el perímetro.
Mochila: Infinito zumbadores
Posición: No importa.
Orientación: No importa.

Entrada



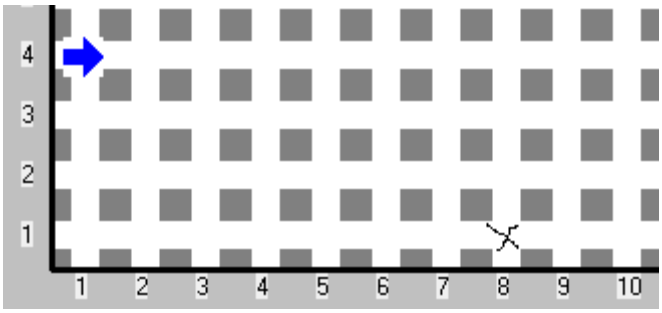
Salida



Ejercicio Recursividad 4: Multiplicando Por Dos

Nuestro querido Karel nos ha visitado en la FAMAT y como resultado aprendio a multiplicar por 2, tristemente perdió el código donde le mostraba como multiplicar. Tu misión es escribir un programa que multiplique por 2

- Karel inicia mirando al Este, en la posición (1,8)
- Karel debe llegar a las coordenadas (28,1), poner un beeper (el único que carga) y termina.
- No hay obstáculos en tu camino
- Antes de correr tu programa asegúrate de ponerle un beeper a Karel en su beeperbag



Ejercicio Recursividad 5: Kuenta Kuantos

Descripción

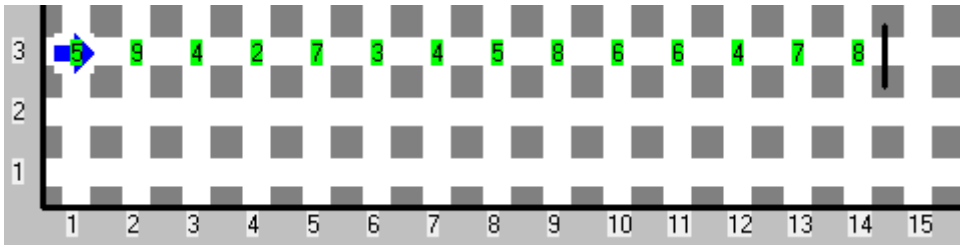
En el mundo de Karel hay un renglón, con varios platos de chicharos (zumbadores). Su tarea es obtener la cantidad de platos que contienen un chicharo, dos chicharos,…,9 chicharos.

Problema

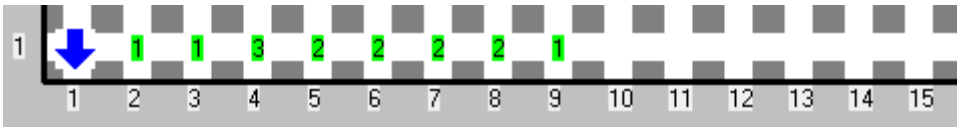
Dada una fila con celdas (platos) que contiene diversas cantidades de zumbadores (chicharos), dar la cantidad de celdas (platos) que contienen, de 0 hasta 9 zumbadores.

Consideraciones

1) Los zumbadores están distribuidos en las celdas del renglón del 3. En cada celda pueden haber desde 0 hasta 99 zumbadores.
2) Todos los zumbadores que hay en el mundo se encuentran en el renglón 3 y están limitados por la derecha por una pared.
3) La única pared en el mundo es la que limita a los zumbadores.
4) No se sabe la posición ni orientación inicial de Karel.
5) Al inicio Karel no tiene zumbadores en la mochila.
6) Al final, en la celda (n,1), n>0, debe haber tantos zumbadores como cantidad de celdas existan con n zumbadores. No importan las otras celas.
7) Karel debe terminar en la celda (1, 1).



Ejemplo de Entrada



Ejemplo de Salida

Ejercicio Recursividad 6: Karel y sus juguetes

Karel dejo tirados sus juguetes, zumbadores, en dos montones distintos del renglón uno y su mama le ordeno que los juntara y pusiera en un cajón que se encuentra exactamente a la misma distancia de los dos montones.
Especificaciones:

Entrada

Mundo: No hay muros. Los zumbadores del mundo se encuentran distribuidos solamente en dos montones. Estos se encuentran en el renglón uno a una distancia impar. La cantidad de zumbadores en cada montón es desconocida.

Mochila: Sin zumbadores.

Posición: Cualquier posición del renglón uno entre los dos montones de zumbadores.

Orientación: Desconocida

Salida

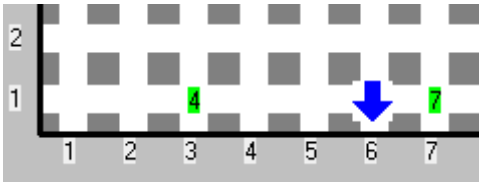
Mundo: Todos los zumbadores del mundo inicial en la casilla que se encuentra exactamente a la misma distancia de la posición original de los dos montones.

Mochila: Sin zumbadores

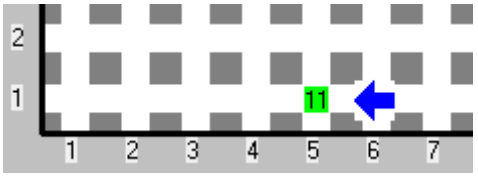
Posición: No importa.

Orientación: No importa.

Entrada



Salida



Ejercicio Recursividad 7: Karelcuates

Karel quiere repartir dulces. Zumbadores, de manera equitativa a sus cuates quedándose el con los que le sobren. Tu tarea es encontrar la cantidad de dulces con los que se quedara Karel.

Especificaciones:

Entrada

Mundo: No hay muros. En la casilla (2.2) estará la cantidad de dulces y en la (3.2) la cantidad de amigos. Karel siempre tiene al menos un amigo.

Mochila: Cien zumbadores.

Posición: (1,1)

Orientación: Este

Salida

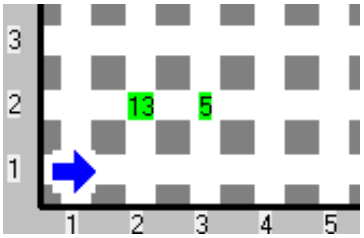
Mundo: La cantidad de dulces con la que se quedara Karel en la posición (4,2). No importa el resto del mundo.

Mochila: No importa

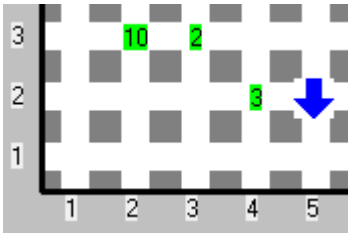
Posición: No importa.

Orientación: No importa

Entrada



Salida



Ejercicio Recursividad 8: Karel Revoruja

Descripción
Karlitos y Karina son hermanos, y como buenos hermanos se la pasan molestándose uno al otro. Karina, que es muy ordenada, apilo todos sus cuadernos (zumbadores), al ver esto, Karlitos los apilo exactamente en orden inverso. Ayuda a Karen a reordenar sus cuadernos

Problema
Poner en orden inverso una lista de zumbadores (cuadernos) limitados por dos paredes.

Consideraciones

1) Los zumbadores están distribuidos en las celdas de una columna de la 2 a la 98. En cada celda puede haber de 0 hasta 99 zumbadores.
2) Los zumbadores están limitados por dos paredes, uno en la parte inferior y otra en la parte superior.
3) Las únicas paredes en el mundo son las que limitan a los zumbadores.
4) Los únicos zumbadores en el mundo son los limitados por las paredes.
5) Karel se encuentra entre las dos paredes que limitan a los zumbadores.
6) Al final los zumbadores deben estar dispuestos en el orden inverso al que se presentaron inicialmente.
7) Karel tiene al inicio dos zumbadores en su mochila.
8) Karel puede terminar en cualquier celda con cualquier orientación, de preferencia, la que más te guste.

Entrada **Salida**

La suma total de zumbadores para será n<=4000

Tarea recursividad 1: La pared

Descripción

Los suegros de Karel siempre le han impedido que él se vea con su novia, a tal grado que edificaron una gran pared de forma irregular para que el no pueda verse con su novia.

Pero como todo buen galán, Karel ha buscado una forma de verse con su novia mediante un pequeño agujerito que ha hecho a la pared que los divide a los dos. Sin embargo Karel debe de decirle a su novia en que parte de la pared se encuentra tal agujero y es por eso que necesita de tu ayuda.

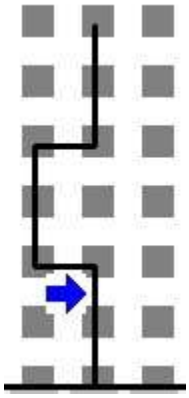
Problema

Ayuda a Karel a mostrarle a su novia en dónde está el agujero que él le hizo a la pared, poniendo un zumbador del otro lado de la pared.

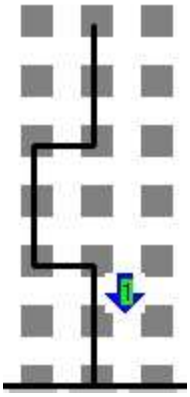
Consideraciones

1. Karel inicia orientado al este en el lugar en donde hizo el agujero.
2. La pared que hicieron los suegros crece de forma irregular, sin embargo solo puede crecer hacia el norte y oeste.
3. Karel únicamente lleva un zumbador en la mochila que será el que pondrá al otro lado de la pared en el lugar en donde se encuentra el agujero.
4. Se asegura que siempre se podrá cruzar al otro lado de la pared
5. No importa la orientación ni la posición final de Karel.

Ejemplo



Mundo de ejemplo



Solución al mundo de ejemplo

Tarea recursiva 2: Travesía escolar

Descripción

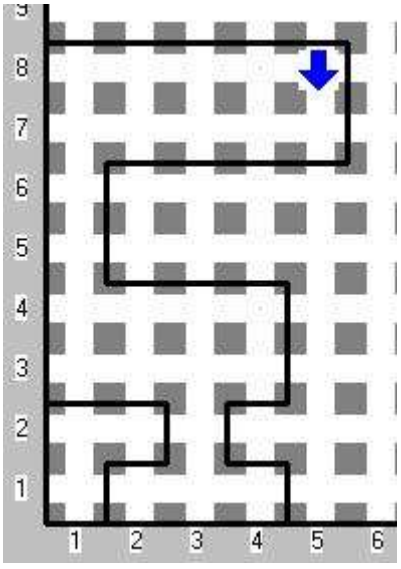
Ahora que Karel ha terminado la secundaria, ha presentado su examen único y para su agrado, ha quedado admitido en uno de los CCH de la Universidad.
Sin embargo hay algo que no le agrada a Karel de la escuela que le tocó. Pese a que su casa está a menos de 100 pasos de distancia de su escuela, el camino que tiene que seguir para llegar a ella, es de lo más tétrico y sinuoso.
Afortunadamente, Karel se siente seguro cuando la longitud del camino es mayor a uno, sin embargo, cuando se ve en la necesidad de pasar por tramos de camino de tamaño uno, se siente atemorizado.

Problema
Dada la forma del camino que Karel tiene que recorrer para ir a la escuela, determina el número de pasos que Karel va a estar encerrado en caminos de tamaño uno.

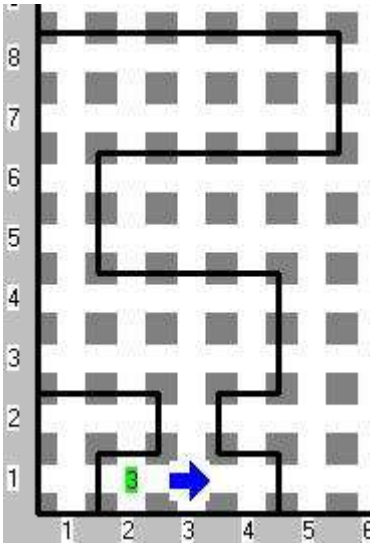
Consideraciones

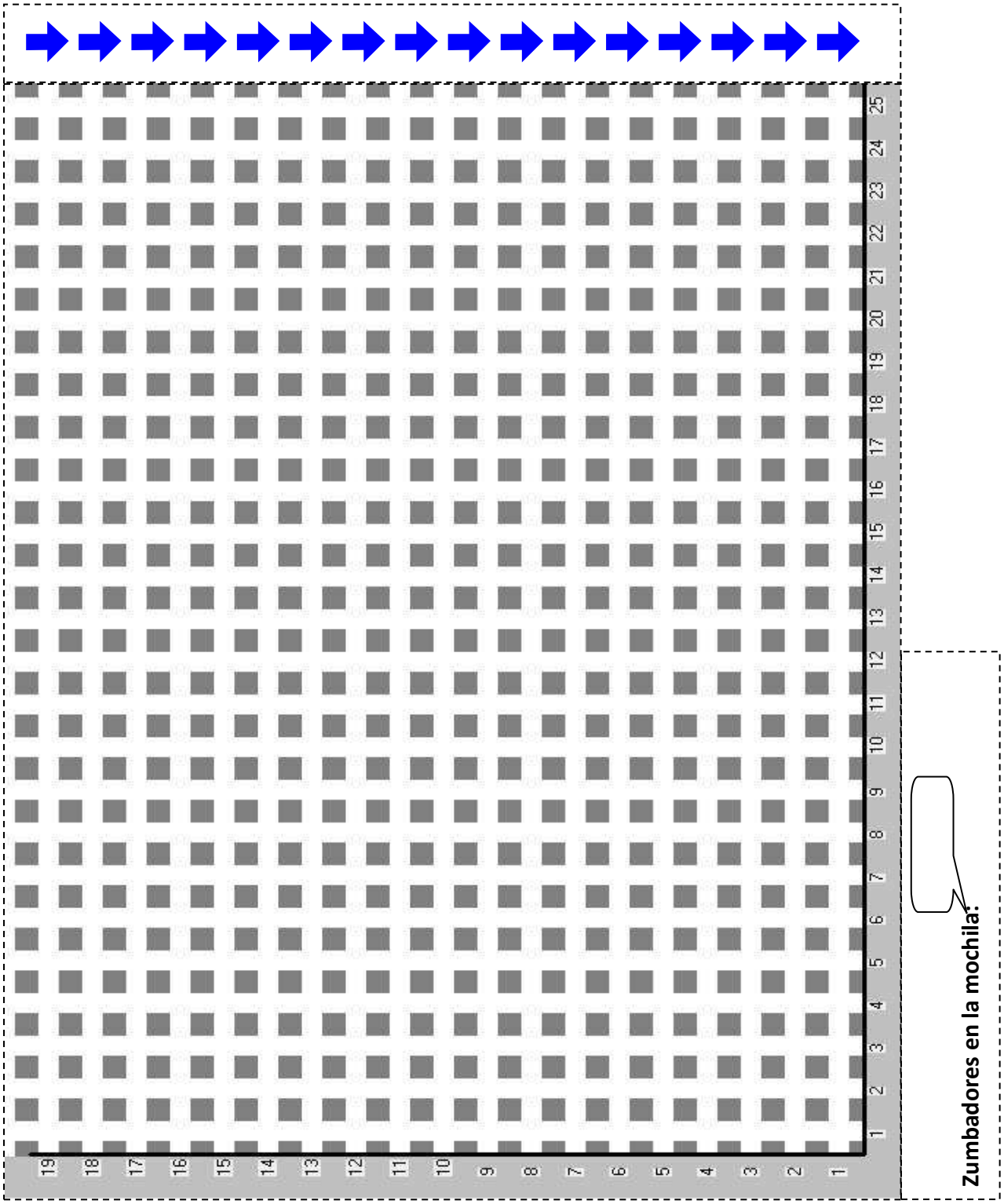
- El camino de la escuela a la casa de Karel está representado de forma vertical donde la pared superior es la casa de Karel y la pared inferior, la escuela.
- Una avenida medida verticalmente, representa un paso que Karel tiene que dar.
- Karel comienza en algún punto de su casa orientado al sur.
- Karel lleva zumbadores infinitos en la mochila.
- Karel debe colocar zumbadores en la esquina inferior izquierda del mapa (donde esta la escuela), representando el número de pasos que Karel tiene que dar en caminos de tamaño uno.
- En ningún punto del camino hay bifurcaciones, es decir, el camino es único.
- No importa la posición ni la orientación final de Karel.
- Ni en la casa, ni en la escuela, el tamaño del camino es de uno.

Mundo Ejemplo



Solución ejemplo




Zumbadores en la mochila:

Notas

Notas