2 INTRUCCIONES PRIMITIVAS DE KAREL

Para que Karel realice una tarea en su mundo, debe dársele un conjunto detallado de instrucciones que le indiquen cómo realizarla. Karel es capaz de recibir, memorizar y ejecutar ese conjunto de instrucciones o programas.

En este capítulo se explican las cinco instrucciones primitivas (vocabulario básico) del lenguaje de Karel, con las que se ordena al robot moverse a través de su mundo y manejar los elementos que allí se encuentran.

2.1. EL VOCABULARIO PRIMITIVO DE KAREL

Las instrucciones primitivas son órdenes que le permite a Karel resolver tareas.

2.1.1 Primitiva cambio de posición: Karel tiene dos instrucciones para cambiar de posición: "**move** ()" cambia su localización y "**turnLeft ()**" su orientación.

Mensaje **move ()**: si el frente está despejado (no hay un muro a media cuadra), se mueve una cuadra en esa dirección. Si hay un muro al frente, a media cuadra, se apaga después de informar el error.
En la figura 33 se observa como karel ejecuta esta instrucción en dos situaciones posibles:

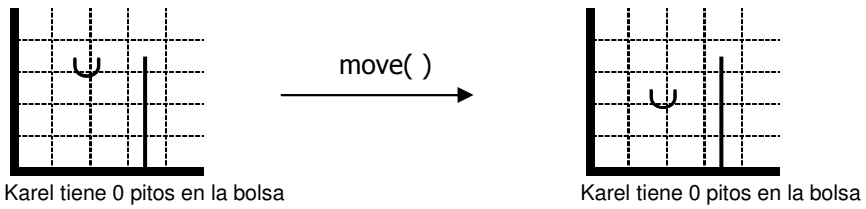


Figura 33 Estado inicial y final de Karel después de ejecutar una instrucción

Puesto que al frente (a media cuadra de distancia) no hay muro, Karel ve el camino despejado y avanza una cuadra en la dirección en que está mirando, en este caso, hacia el sur.

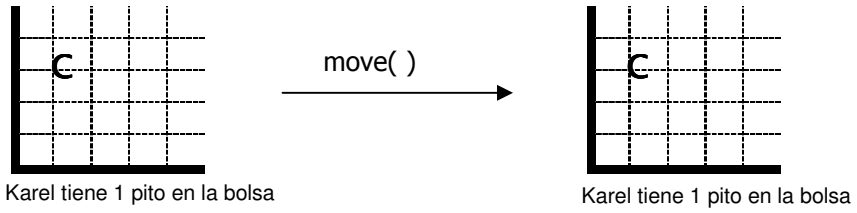


Figura 34 Estado inicial y final de Karel después de ejecutar una instrucción

Con su cámara delantera Karel ve un muro a media cuadra de distancia y como medida de seguridad no ejecuta el movimiento que lo haría estrellarse contra éste, sino que, se apaga después de informar el error. Esta acción de apagarse debido a un error del programa la llamaremos apagarse por error (error shutOff)

Mensaje turnLeft () (Giro a la izquierda): Karel ejecuta esta instrucción rotando 90 grados hacia su izquierda. Los otros elementos del mundo (muros y pitos) y la localización de Karel no se modifican. Lo único que cambia es su orientación. Karel nunca se apaga por error al ejecutar la instrucción girar hacia la izquierda (siempre puede realizar el giro). En la figura 35 se muestran cuatro situaciones posibles en las que Karel puede ejecutar esta instrucción y sus respectivos resultados:

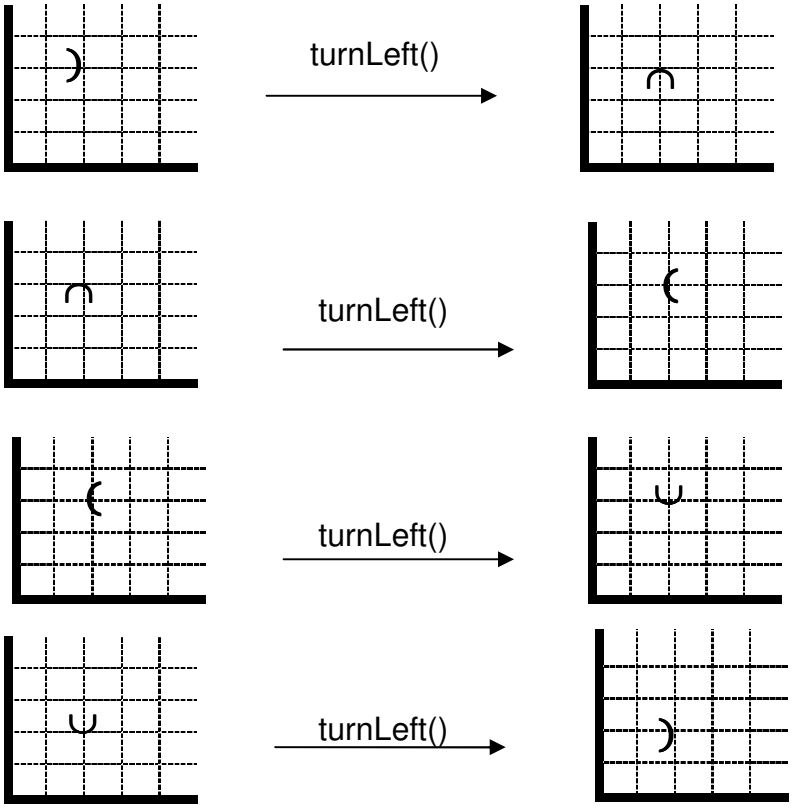


Figura. 35 Estados posibles de karel después de ejecutar la instrucción turnLeft

2.1.2 Manejo de Pitos: Karel puede recoger los pitos de su esquina, llevarlos en su bolsa y ponerlos en otra esquina. Para recoger y guardar un pito en la bolsa tenemos la instrucción "**pickBeeper** ()" y para tomar un pito de la bolsa y ponerlo en la esquina usamos "**putBeeper** ()"

Mensaje pickBeeper (): si hay pitos en su esquina, Karel toma uno y lo guarda en su bolsa. Si no hay pitos en su esquina, el robot se apaga por error. Estos dos casos se ilustran en la figura 36:

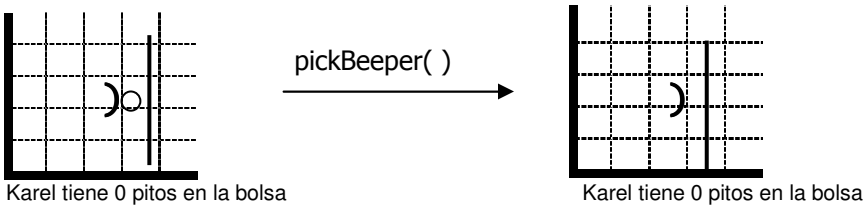


Figura. 36 Estados posibles después de ejecutar la instrucción pickBeeper

Karel oye el sonido de los pitos en su esquina, recoge uno de ellos y lo guarda en la bolsa. En la figura 37 se observa que el número total de pitos sigue siendo uno, pues ahora hay cero pitos en la esquina y uno en la bolsa de Karel. Como no hay pitos en la esquina de Karel, éste se apaga por error sin realizar otra acción.

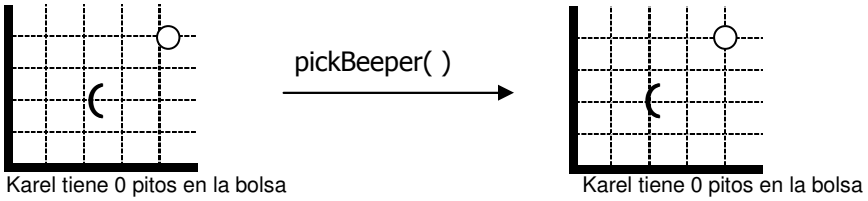


Figura. 37 Estados posibles de Karel después de ejecutar la instrucción pickBeeper

Mensaje putBeeper (): si tiene pitos en la bolsa, saca uno y lo pone en su esquina y si tienen la bolsa vacía se apaga por error.

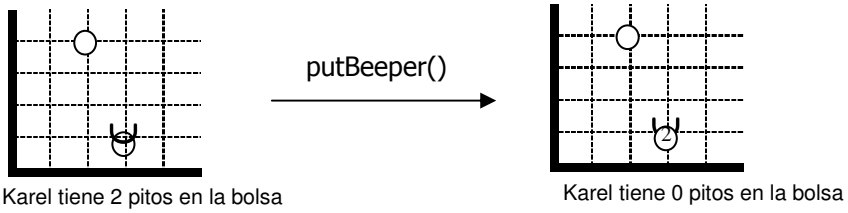


Figura. 38 Estados posibles de Karel después de ejecutar la instrucción putBeeper

Karel realiza el putBeeper () revisando en su bolsa si tiene pitos, coge uno y lo coloca en la esquina en la que está; en la situación final quedan dos pitos en la esquina y uno en la bolsa. Como no tiene pitos en su bolsa, Karel se apaga por error.

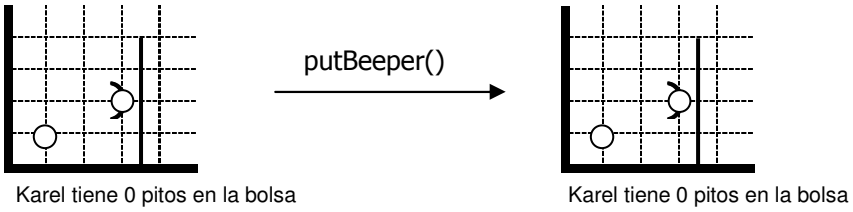


Figura. 39 Estados posibles de Karel después de ejecutar la instrucción putBeeper

2.1.3 Terminación de instrucciones

Mensaje turnOff (): Karel se apaga normalmente, sin errores. Esta instrucción debe ser siempre la última que ejecute, cuando ya haya llegado a la situación final del problema a resolver, se debe colocar dentro de un programa como la última instrucción.

2.2 SOLUCIÓN DE PROBLEMAS

Las instrucciones primitivas resuelven problemas. Para resolver esta clase de problema basta con saber qué instrucción le permitirá a Karel transformar el estado inicial en el estado final.

Problema 1. En las figuras 40 y 41. ¿Qué primitivas utilizaría karel? Para resolver los problemas

Primer caso:

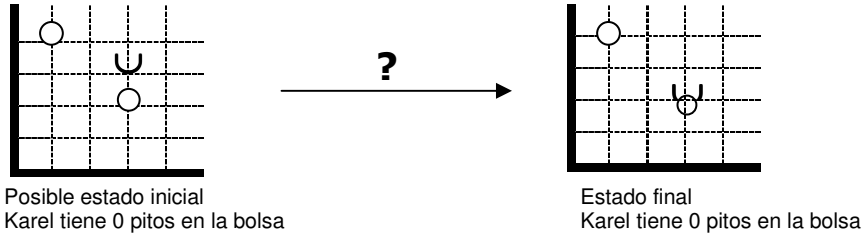


Figura. 40 Tipo de instrucción que le permite a Karel transformar el estado inicial en estado final

Segundo caso:

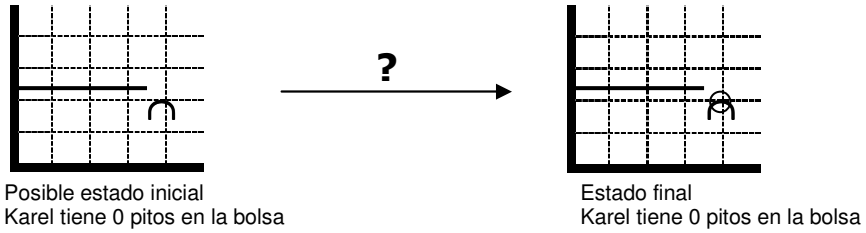


Figura. 41 Tipo de instrucción que le permite a Karel transformar el estado inicial en estado final

Soluciones:

Primer caso: En el primer caso, lo único que cambia es la posición de Karel. Sólo se necesita una instrucción move ( ), figura 42.

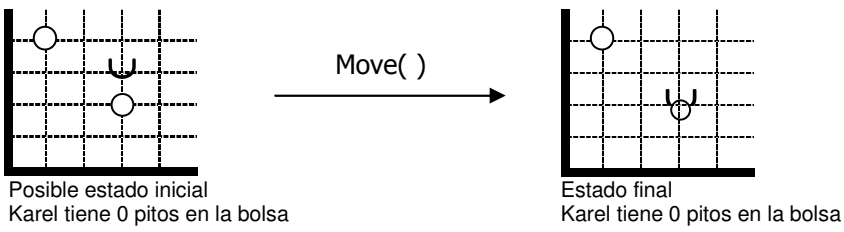


Figura. 42 Uso de la instrucción move para la solución del primer caso

Segundo caso:

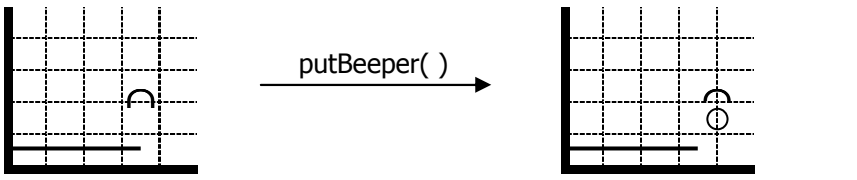


Figura. 43 Uso de la instrucción putBeeper para la solución del segundo caso

Cuando existen tareas más complejas que karel no puede resolver con una sola instrucción primitiva, se debe llevar a cabo el siguiente proceso:

1. Identificar uno o más estados intermedios para partir el problema en problemas más simples (subproblemas).

2. Resolver cada uno de los subproblemas simples. Para los complicados es necesario aplicar de nuevo todo el proceso.

3. Unir las soluciones de los subproblemas para conformar la solución global del problema. En el lenguaje de Karel se utiliza el signo **';'** (punto y coma) para componer (unir) las soluciones de dos subproblemas.

Problema 2: En la figura 44 se le ordena a karel resolver la situación presentada:

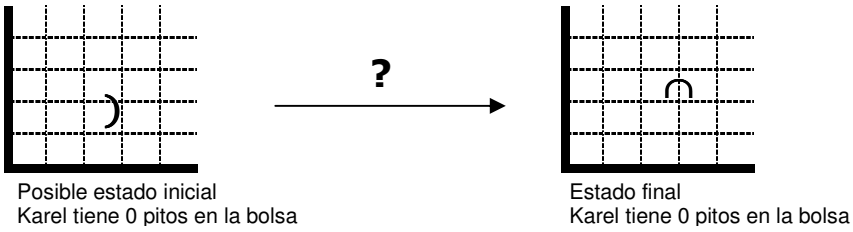


Figura. 44 Estados posibles para el problema 2

Solución:

Se deben identificar los estados intermedios del problema para obtener así dos subproblemas:

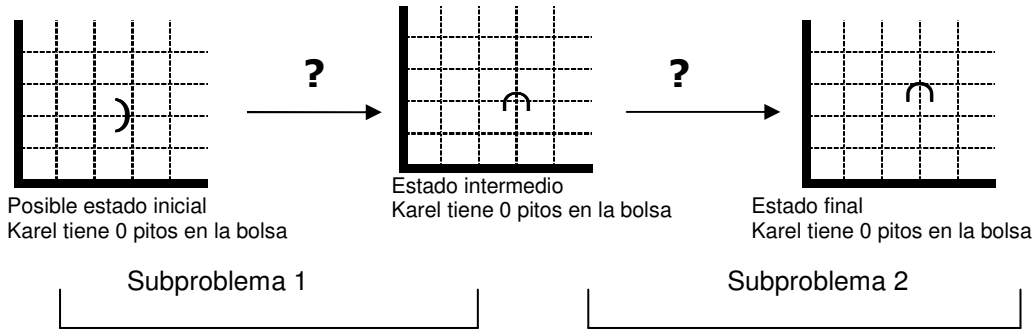


Figura 45. Estados Intermedios para el problema 2

Ahora tenemos dos subproblemas simples que se pueden resolver directamente utilizando instrucciones primitivas. Finalmente se deben unir las soluciones mediante un ';'(punto y coma), obteniendo un programa: El cual representa la solución del problema planteado.

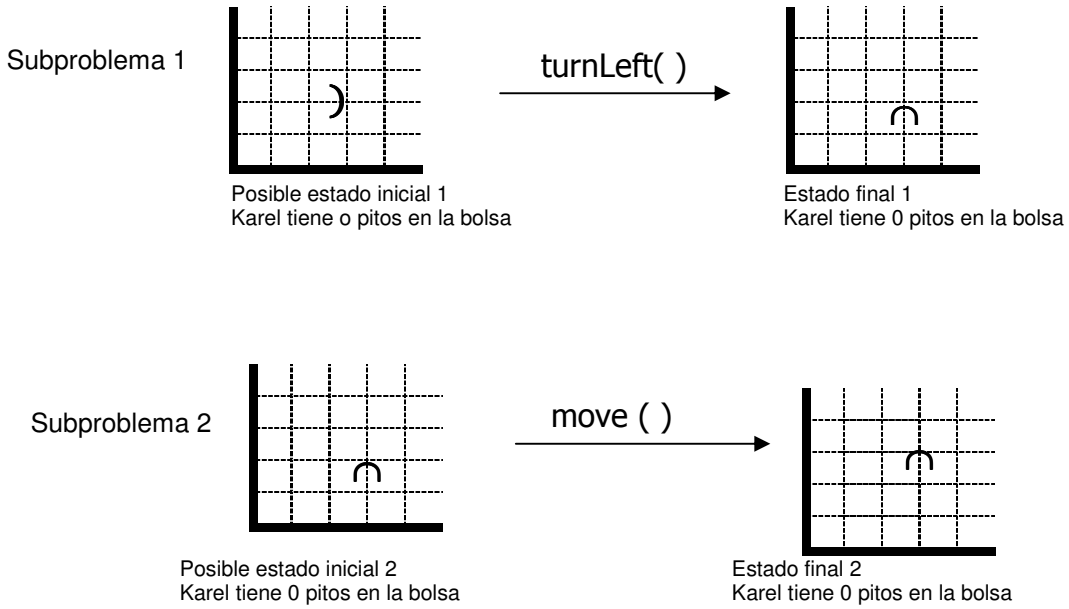


Figura 46 Estados inicial y final de los subproblemas 1 y 2 del problema 2

En algunos casos es necesario identificar varios estados intermedios tal como se ilustra en la figura 47:

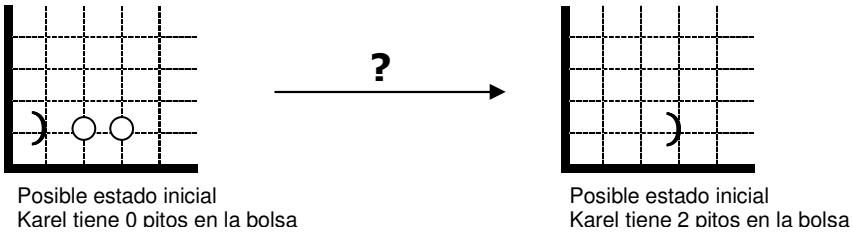


Figura. 47 Posibles estados iniciales y finales

Solución:

Se divide el problema en cuatro subproblemas y se resuelve cada uno independientemente, como se observa en la figura 48.

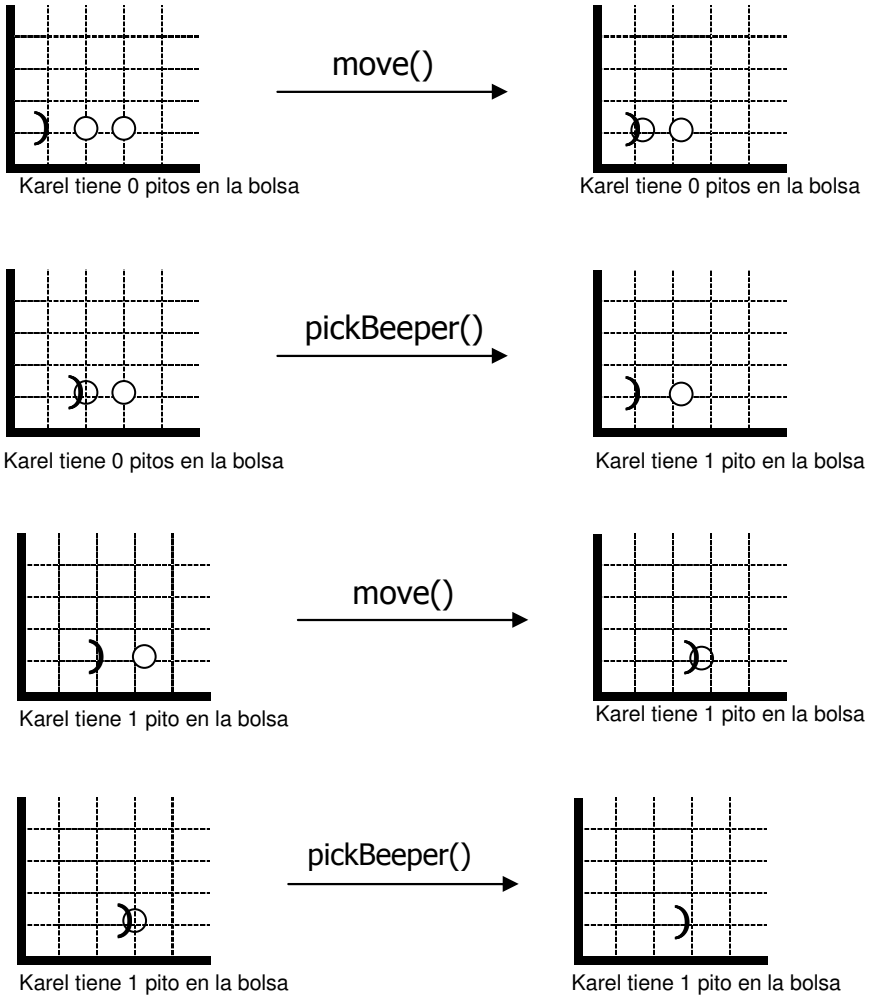


Figura. 48 Posibles estados iniciales y finales

2.3 ESTRUCTURA DE UN PROGRAMA COMPLETO

Hasta el momento se han explicado las instrucciones primitivas del lenguaje de Karel y su uso para resolver problemas, A continuación se le define la forma de comunicarle a Karel las instrucciones que conforman la solución a un problema, de manera que las entienda y pueda ejecutar el programa.

El lenguaje de Karel emplea el **';'** (punto y coma) y un conjunto de palabras reservadas que sirven para estructurar el programa, convencionalmente estas palabras se escriben en mayúsculas y las instrucciones en minúsculas.

2.3.1 Codificación: La codificación de los programas consiste en asignar la sintaxis correcta a las instrucciones para que karel obedezca las órdenes.

Ejemplo de la codificación de un programa:

```
    task
{
    ur_robot karel(2,3,East,0);
    karel.turnLeft();
    karel.move();
    karel.turnOff();
}
```

```
task
{
    ur_robot karel(1,1,East,0);
    karel.move();
    karel.pickBeeper();
    karel.move();
    karel.pickBeeper();
    karel.turnOff()
   }
```

2.3.2 Ejecución de un programa: Para que Karel ejecute un programa son necesarios los siguientes pasos:

Prenderlo: Karel queda en modo lectura, listo para "oír" el programa mediante el mensaje ur_robot karel(n,m,o,k) donde n es el número de la calle, m es el número de la avenida, o es la orientación del robot y k el número de pitos en la bolsa.

Bloque del programa {…}: Encierra todo el programa que le vamos a dar a Karel.

Dentro del bloque {…}: Colocamos, una tras otra, todas las instrucciones del programa, incluyendo al final la instrucción turnOff() para que Karel se apague al terminar.

Pares de símbolos { }: Limitan el comienzo y final de un grupo de instrucciones y se llaman delimitadores.

2.4 ERRORES DE PROGRAMACIÓN

Los errores de programación se clasifican en cuatro categorías:

2.4.1 Errores Léxicos: Ocurren cuando Karel encuentra una palabra que no está en su vocabulario.

2.4.2 Sintácticos: tienen que ver con la "gramática" y la "puntuación" del programa, es decir, con las palabras reservadas y la localización del ';' (punto y coma). Karel detecta estos dos tipos de errores en el modo de lectura; cuando encuentra un error lo notifica y se apaga.

2.4.3 De ejecución: Si Karel entiende el programa (no hay errores léxicos ni sintácticos) se prende el botón de ejecución y comienza la ejecución del programa; en este modo pueden presentarse errores cuando Karel no puede realizar una primitiva correctamente y se apaga por error (move() con el frente bloqueado, sin pickBeeper() cuando no hay pitos en la esquina, putBeeper() sin pitos en la bolsa). Estos son los errores de ejecución.

2.4.4 De intención: Los errores de intención, que son los más difíciles de identificar, ocurren cuando Karel ejecuta correctamente el programa, pero éste no resuelve la tarea especificada (partiendo del estado inicial dado, no se llega al estado final deseado).

2.5 CASO PRÁCTICO

A continuación se especifica una tarea en términos de un estado inicial y su estado final y se presenta un programa escrito para solucionarlo.

Revise el programa, identifique y corrija los errores léxicos y sintácticos que encuentre. Indique los errores de ejecución y modifique el programa para que resuelva la tarea dada (hemos enumerado las líneas del programa únicamente para facilitar el ejercicio).

Problema Caso Práctico: Karel se encuentra en el origen mirando al Este y debe recoger el pito que se encuentra en el origen, el que se encuentra dos calles adelante y regresar al origen: Véase figura 49

Figura 49 Estado inicial y final del problema del caso práctico

Solución:

```
task;
{
3    ur_robot karel (1, 1, East, 0);
4     karel.pickBeeper ();
5     karel.move (); karel.pickBeeper ();
6     karel.move (); karel.pickBeeper ()
7     karel.turnLeft ();
8     karel.move ();
9     move ();
10   }
```

Errores:

- **Léxicos**

  línea 5 karel.pickBeeper();

  línea 7 karel.turnLeft();

- **Sintácticos:**

  línea 1 **;**

  línea 6 falta ;

  línea 8 **; ;**

línea 9 Falta referencia objeto karel

- **De ejecución.**

línea 4 al ejecutar karel.pickBeeper();

línea 10 falta antes de } karel.turnOff()

2.6 EJERCICIOS PROPUESTOS

Ejercicio. 1 ¿Cómo ejecuta Karel la instrucción move en los siguientes estados (dibuje el estado final)?

a)

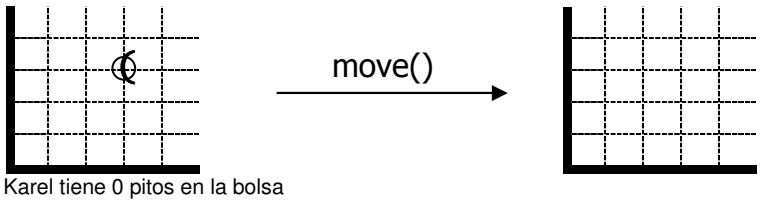


Figura. 50 Posibles estados inicial y final de la instrucción move()

b)

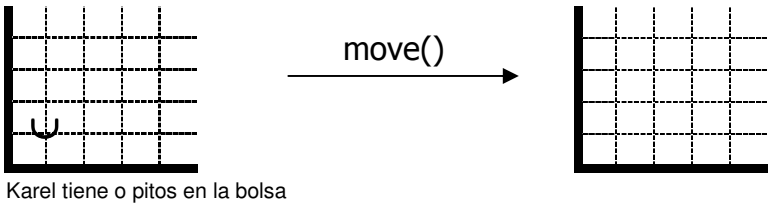


Figura. 51 Posibles estados inicial y final de la instrucción move()

c)

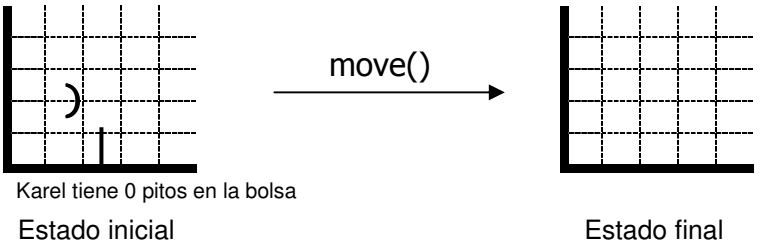


Figura. 52 Posibles estados inicial y final de la instrucción move()

d)

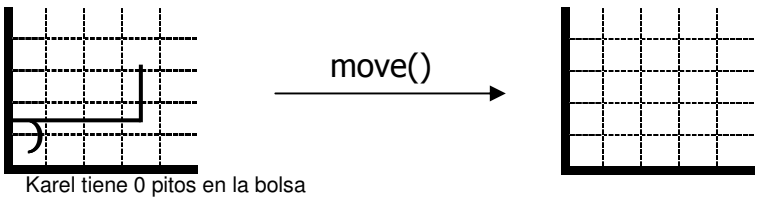


Figura. 53 Posibles estados inicial y final de la instrucción move()

Ejercicio. 2 ¿Cómo quedaría el mundo de Karel después de realizar un turnLeft()?

a)

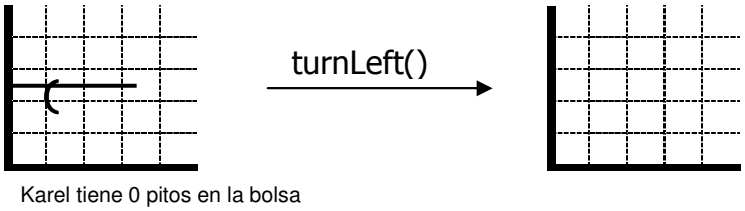


Figura. 54 Posibles estados inicial y final de la instrucción turnLeft()

b)

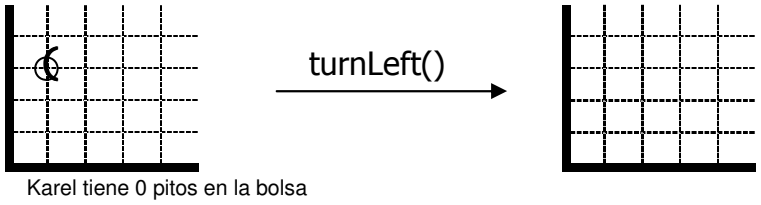


Figura. 55 Posibles estados inicial y final de la instrucción turnLeft()

c)

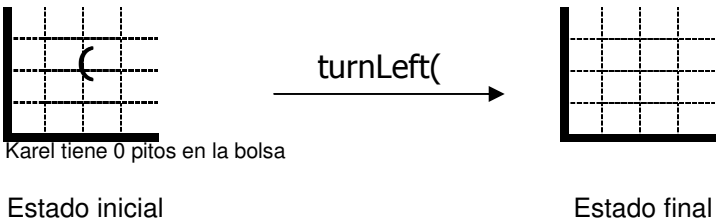


Figura. 56 Posibles estados inicial y final de la instrucción turnLeft()

d)

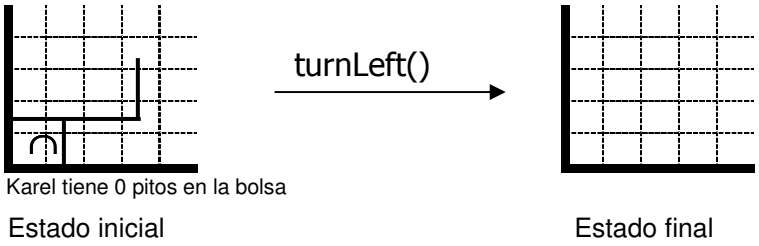


Figura. 57 Posibles estados inicial y final de la instrucción turnLeft()

Ejercicio. 3 ¿Cómo ejecuta Karel pickBeeper() en los siguientes estados?

a)

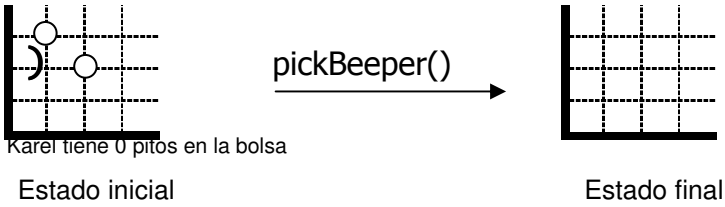


Figura. 58 Posibles estados inicial y final de la instrucción pickBeeper()

b)

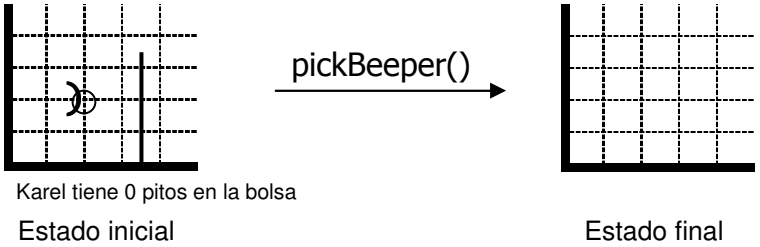


Figura. 59 Posibles estados inicial y final de la instrucción pickBeeper()

c)

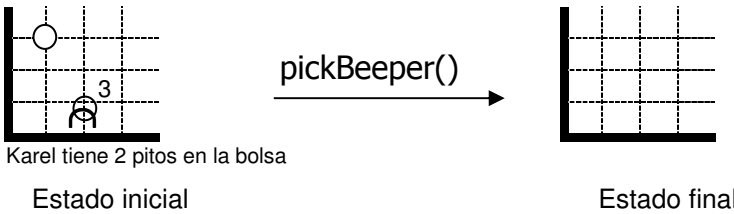


Figura. 60 Posibles estados inicial y final de la instrucción pickBeeper()

d)

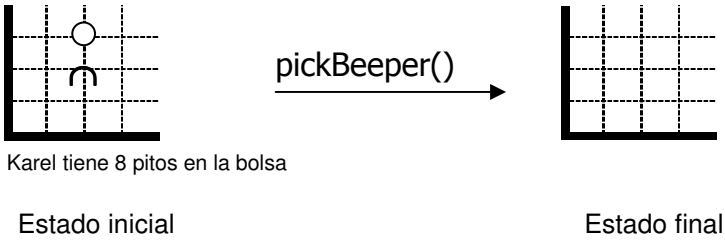


Figura. 61 Posibles estados inicial y final de la instrucción pickBeeper()

Ejercicio. 4 ¿Cómo ejecuta Karel putBeeper() en los siguientes estados?

a)

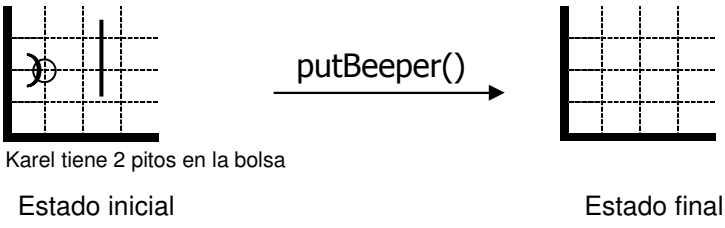


Figura. 62 Posibles estados inicial y final de la instrucción putBeeper()

b)

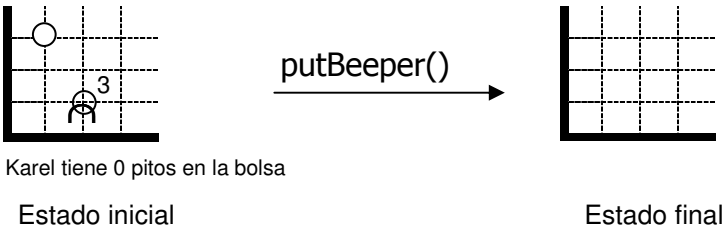


Figura. 63 Posibles estados inicial y final de la instrucción putBeeper()

c)

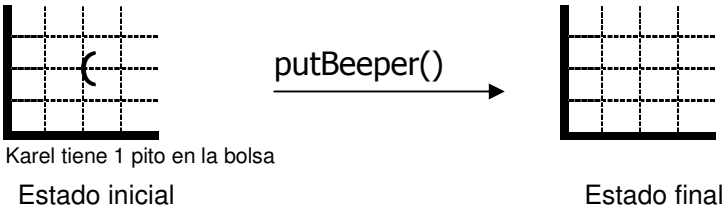


Figura. 64 Posibles estados inicial y final de la instrucción putBeeper()

d)

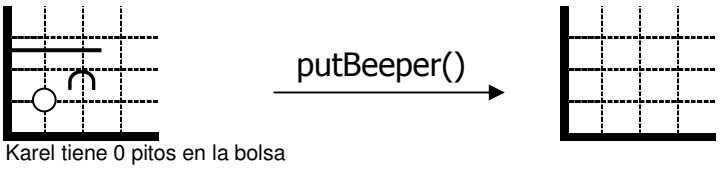


Figura. 65 Posibles estados inicial y final de la instrucción putBeeper()

Ejercicio. 5 ¿Determine las primitivas que se deben utilizar para resolver los siguientes problemas?.

a)

Figura. 66 Posibles estados inicial y final para el uso de primitivas

b)

Figura. 67 Posibles estados inicial y final para el uso de primitivas

c)

Figura. 68 Posibles estados inicial y final para el uso de primitivas

Ejercicio. 6 Resuelva los siguientes problemas identificando sus estados intermedios.

a)

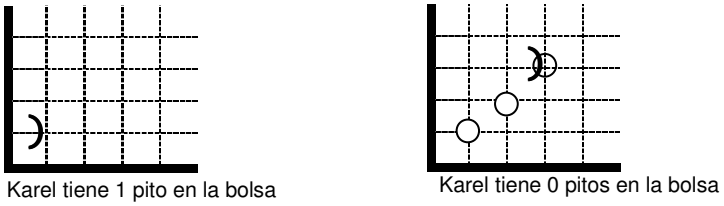


Figura. 69 Posibles estados inicial y final para identificar los intermedios

b)

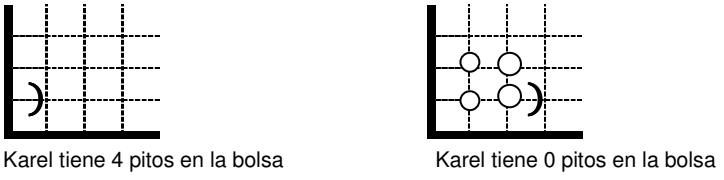


Figura. 70 Posibles estados inicial y final para identificar los intermedios

c)

Figura. 71 Posibles estados inicial y final para identificar los intermedios

d)

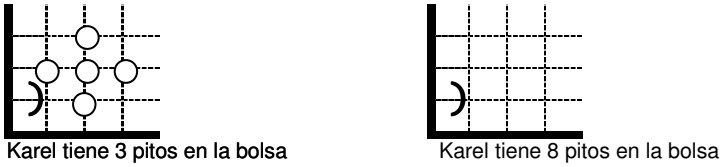
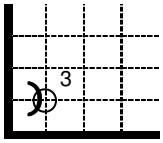


Figura. 72 Posibles estados inicial y final para identificar los intermedios

Ejercicio. 7 Identifique los errores de programación que aparecen en cada uno de los siguientes programas.

a)



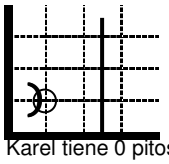
Karel tiene 0 pitos en la bolsa

Estado inicial

Figura. 73 Posible estado inicial para identificar los errores de programación

```
task
{
ur_robot karel(1,1,East,0);
karel.pickBeeper();
karel.pickBeeper();
2 karel.move();
karel.turnLeft();
karel.turnOff();
}
```

b)



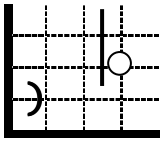
Karel tiene 0 pitos en la bolsa

Estado inicial

Figura. 74 Estado inicial para identificar los errores de programación

```
task
{
ur_robot karel(1,1,East,0);
karel.pickBeeper();
karel.putBeeper();
karel.move();
karel.move();
karel.turnRight();
karel.turnOff() ;
}
```

c)



Karel tiene 0 pitos en la bolsa

Figura. 75 Estado inicial para identificar los errores de programación

```
task
{
ur_robot karel(1,1,East,0);
karel.move();
karel.move();
karel.turnLeft();
karel.move();
karel.pickBeeper();
karel.turnLeft();
karel.move();
karel.turnOff() ;
}
```