Lenguaje de Programación:

Extendiendo el lenguaje de Karel

José Luis Alonzo Velázquez

Universidad de Guanajuato

Sesión 4

Creando más palabras en el vocabulario de Karel

El tamaño de nuestro código crece a medida que le pedimos a Karel que realice tareas más complicadas. Es común que en un mismo programa tengamos que escribir un mismo código "pedazo de código" en diferentes partes. Pensando en ese problema, el Dr. Isaac le dio a Karel la habilidad de extender su vocabulario. Es decir, podemos enseñarle nuevas instrucciones que ejecutan instrucciones más simples.

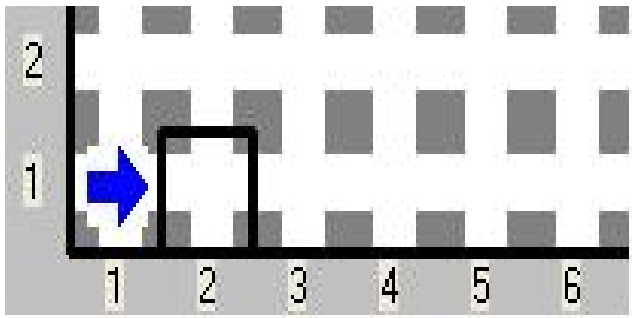
turnright

Imaginemos que deseamos que Karel gire a la derecha, entonces tendríamos que indicarle que diera tres giros a la izquierda. Eso noes tanto problema si sólo necesitáramos una vez girar a la derecha. Sin embargo, si en nuestro código empleamos muchas veces el giro a la derecha, tendríamos que insertar el mismo código por cada giro.

Por lo tanto, es claro que una instrucción **turnright** nos sería de gran utilidad!!!

Subiendo una escalera

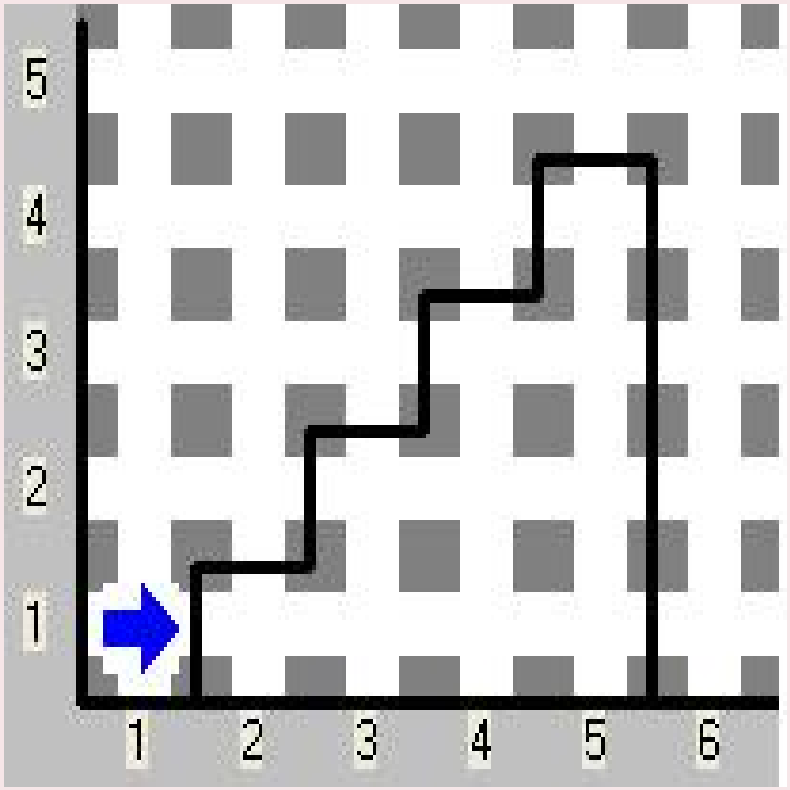
Ahora imaginemos que Karel debe subir el siguiente escalón:



El código para subir un escalón sería:

```
class program{
        program(){
                turnleft();
                move();
                turnleft();
                turnleft();
                turnleft();
                move();
                turnoff();
        }
}
```

No es difícil imaginar lo inmenso del código para subir:



Código

```
class program
      {
       program()
            {
             turnleft();
             move();
             turnleft();
             turnleft();
             turnleft();
             move();
             turnleft();
             move();
             turnleft();
             turnleft();
             turnleft();
             move();
```

continuación del código

```
        turnleft();
        move();
        turnleft();
        turnleft();
        turnleft();
        move();
        turnleft();
        move();
        turnleft();
        turnleft();
        turnleft();
        move();
        turnoff();
        }
}
```

Para definir una nueva instrucción

define <nombre_nueva_instrucción>()
{
<instrucción>
<instrucción>
⋮
<instrucción>
}

La palabra reservada **define** la empleamos para decirle a Karel que le enseñaremos una nueva instrucción. Posteriormente viene el nombre de la nueva instrucción, la cual no debe ser igual al nombre de alguna otra instrucción nativa o ya definida(tampoco palabras reservadas de Karel, es decir aquellas que no son instrucciones pero tienen alguna función en el lenguaje de programación).

Bloque de instrucciones

La instrucción puede ser una sola instrucción o un bloque de instrucciones creado con "{" y "}". Esto crea lo que se conoce como una entrada del diccionario de Karel, que es donde está todo lo que Karel puede y sabe ejecutar. Es una buena costumbre de programación darles nombres a las instrucciones de acuerdo a la acción que realizan, lo cual simplificará poder entender los programas que escribimos.

IMPORTANTE

Recordar que no se pueden usar nombres de instrucciones nativas ni palabras reservadas. Karel hace distinción entre mayúsculas y minúsculas. Tampoco se puede usar caracteres especiales.

Definición de la función de giro a la derecha.

```
define turnright()
        {
        turnleft();
        turnleft();
        turnleft();
        }
```

Definición de la instrucción sube_escalon.

```
define sube_escalon()
        {
        turnleft();
        move();
        turnright();
        move();
        }
```

Con lo cual nuestro programa para subir las escaleras sería:

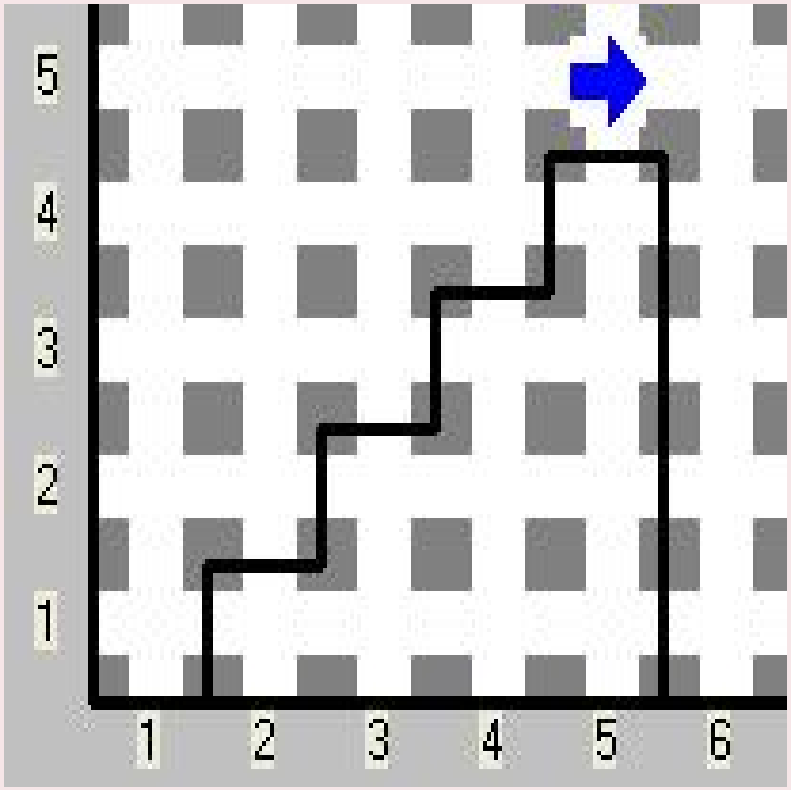
```
class program
{
define turnright()
        {
        turnleft();
        turnleft();
        turnleft();
        }
define sube_escalon()
        {
        turnleft();
        move();
        turnright();
        move();
        }
```

Continuación del codigo

```
        program()
        {
        sube_escalon();
        sube_escalon();
        sube_escalon();
        turnoff();
        }
}
```

Notar que las nuevas instrucciones se encuentran definidas dentro de **class program** y van antes de la función principal llamada **program**. Es importante resaltar el uso de nuevas instrucciones dentro de nuevas instrucciones. En nuestro caso, la instrucción **turnright** se emplea dentro de la instrucción **sube_escalon**.

Resultado



Razones para crear nuevas instrucciones

Siempre se puede escribir un programa sin instrucciones nuevas, pero resulta una larga secuencia de comandos básicos de Karel, y es muy complicado de entender, corregir y modificar. Es un buen hábito de programación, el tener nuevas instrucciones definidas en la forma más clara y concisa posible, para poder reutilizarlas en futuros programas.

Incluso es posible tener una biblioteca de múltiples funciones de uso variado, para agregarlas dentro de los nuevos programas escritos.

Aplicación sencilla

Una aplicación sencilla es renombrar las instrucciones nativas(move, putbeeper, etc.), para tenerlas de manera más clara y evitar errores a la hora de escribirlas. Por ejemplo:

```
define pon_biper()
        {
        putbeeper();
        }
```

La claridad en un código nos ayuda a comprender que estamos haciendo y cómo estamos dividiendo nuestra tarea en tareas más pequeñas.

Programa poco claro

```
program()
        {
        tr();
        move();
        pon();
        c();
        pon();
        c();
        rb();
        }
```

Programa entendible

```
program()
        {
        inicia_posicion();
        move();
        coloca_pila();
        cambia_de_avenida();
        coloca_pila();
        cambia_de_avenida();
        recoge_beepers();
        }
```

Problema Karel Armstrong

El ciclista Karel Armstrong esta entrenando para su siguiente competencia. Por lo tanto a dispuesto la pista de entrenamiento como se muestra en la figura. Tu debes indicarle a Armstrong como correr la pista esquivando las paredes que se encuentran en el camino. Tu carrera debe hacerse lo más pegado al piso y detenerte en la avenida 9 tal y como se muestra en la figura. No importa con que orientación termine Karel.

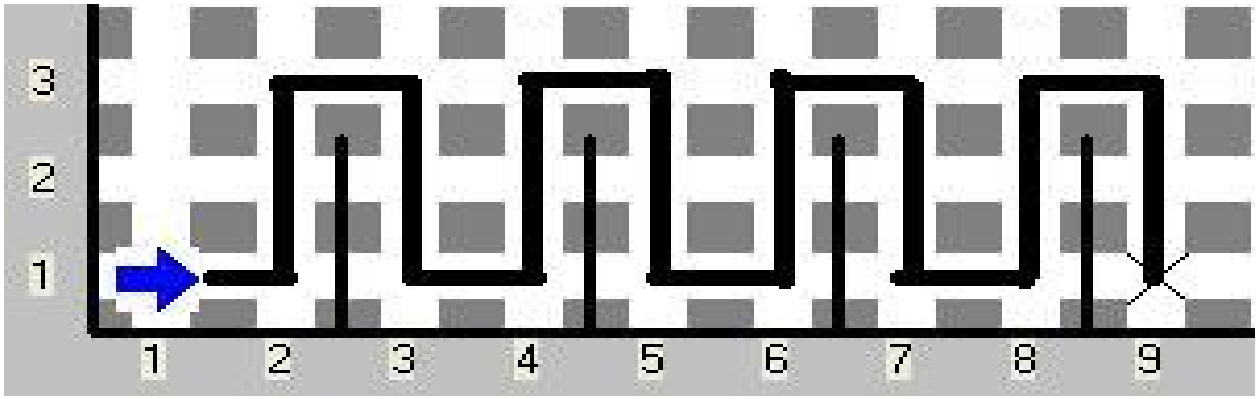


Figura: El recorrido de La Tour de Krance hecho por Karel Armstrong

Solución Karel Armstrong

Notemos que se tienen que saltar paredes de la misma altura, por lo que podemos definir una nueva instrucción que realice la tarea. La instrucción debe pasar la pared y dejar a Karel exactamente mirando al este. Entonces, esto lo podemos lograr haciendo que Karel avance, gire ala izquierda, se mueva dos lugares, luego gire a la derecha, avance un espacio, gire nuevamente a la derecha, avance dos y por último, gire a la izquierda para que mire al este.

La instrucción quedaría de la siguiente manera:

```
define salta_pared()
        {
        move();
        turnleft();
        move();
        move();
        turnright();
        move();
        turnright();
        move();
        move();
        turnleft();
        }
```

Con lo cual el programa sería:

```
program()
        {
        salta_pared();
        salta_pared();
        salta_pared();
        salta_pared();
        turnoff();
        }
```

Edgar Alfredo Duéñez Guzmán & Edgar Said Hernández Sánchez & Marte Alejandro Ramírez Ortegón *Los Dilemas de Karel*. CIMAT, Mayo 2006.