Convenciones

< > Lo que esté entre signos de menor que y mayor que, es una aclaración de lo que debe ir en ese lugar.
[ ] Indica una porción de código que es optativa, o una opción a escoger entre dos.

1. Estructura de programa

Un programa de Karel debe tener el siguiente formato:

```
class program
        {
        <Definiciones de funciones>
        program()
                {
                <Definiciones de las instrucciones a ejecutar>
                turnoff();
                }
        }
```

2. Funciones básicas (nativas)

- **move()**: hace un paso en la dirección que está apuntando (puede causar error "apagón" si hay una pared enfrente).
- **turnleft()**: gira a su izquierda 90º (siempre se puede).
- **pickbeeper():** recoge un beeper de la esquina donde está parado (puede causar error "apagón" si no hay ningún beeper en la esquina).
- **putbeeper():** deposita un beeper en la esquina (puede causar error "apagón" si la bolsa de beepers de Karel está vacía).
- **turnoff():** es el comando que finaliza y apaga a Karel.

IMPORTANTE: Las nuevas funciones deben tener diferentes nombres y NO llamarse igual a una de las funciones básicas.

3. La instrucción iterate

Su sintaxis es la siguiente:

```
iterate (<número>)

<instrucción>[{Bloque}]
```

Por ejemplo:

```
iterate(3)
        turnleft();
iterate(4)
        {
        turnleft();
        move();
        }
```

4. Definir nuevas funciones de Karel (define)

La sintaxis es:

```
define <nombre_nueva_instrucción>()
       <instrucción> [{bloque}]
```

Donde la *instrucción* puede ser una sola instrucción o un bloque de instrucciones creado con { y }.

No se puede usar nombres de instrucciones nativas ni palabras reservadas.
Karel hace distinción entre mayúsculas y minúsculas

```
define turnright()
       {
       iterate(3)
              turnleft();
       }
```

5. Condicionales

Las condiciones posibles son las siguientes 18:

frontIsClear	es verdadero si el frente de Karel esta libre de pared
frontIsBlocked	es verdadero si existe una pared frente a Karel
leftIsClear	es verdadero si en el lado izquierdo de Karel no existe una pared
leftIsBlocked	es verdadero si en el lado izquierdo de Karel existe una pared
rightIsClear	es verdadero si en el lado derecho de Karel no existe una pared
rightIsBlocked	es verdadera si en el lado derecho de Karel existe una pared
nextToABeeper	es verdadero si existe un beeper donde Karel esta parado
notNextToABeeper	es verdadero si no existe un beeper donde Karel esta parado
facingNorth	es verdadero si Karel esta volteando hacia al Norte
notFacingNorth	es verdadero si Karel no esta volteando hacia al Norte
facingSouth	es verdadero si Karel esta volteando hacia el Sur
notFacingSouth	es verdadero si Karel no esta volteando hacia el Sur
facingEast	es verdadero si Karel esta volteando hacia el Este
notFacingEast	es verdadero si Karel no esta volteando hacia el Este
facingWest	es verdadero si Karel esta volteando hacia al Oeste
notFacingWest	es verdadero si Karel no esta volteando hacia al Oeste
anyBeepersInBeeperBag	es verdadero si existe algún beeper en a beeper_bag
noBeepersInBeeperBag	es verdadero si no existes algún beepers en a beeper_bag

6. La instrucción if

Su sintaxis es la siguiente:

```
if( <condición> [bloque condicional] )
        <instrucción> [{bloque}]
```

Por ejemplo:

```
if( frontIsClear )
        move();
if( leftIsClear )
        {
        turnleft();
        move();
        }
```

7. La instrucción else

Su sintaxis es:

```
if( <condición> [bloque condicional] )
        <instrucción1> [{bloque1}]
else
        <instrucción2> [{bloque2}]
```

Por ejemplo:

```
if( frontIsClear )
        move();
else
        turnleft();
if( nextToABeeper )
        {
        pickbeeper();
        move();
        }
else
        putbeeper();
```

En este caso, lo que ocurre es que si la condición es cierta, se ejecuta la instrucción1 (o el bloque1), pero si no lo es se ejecuta la instrucción2 (o el bloque2), e independientemente del resultado, se continua la ejecución del programa.

8. La instrucción while

El **while** tiene la siguiente sintaxis:

```
while( <condición> [bloque condicional] )
        <instrucción> [{bloque}]
```

Por ejemplo:

```
while( frontIsClear )
        move();
while( leftIsClear )
        {
        turnleft();
        move();
        }
```

9. Bloques condicionales

Los bloques condicionales, son un grupo de condiciones que se piden en una instrucción **if** o **while**. Las condiciones deben ir "ligadas" con un operador lógico "and" u "or". La sintaxis del *and* y *or* son respectivamente:

- And: <*Condición 1*> && <*Condición 2*>
- Or: <*Condición 1*> || <*Condición 2*>

Por ejemplo, para *and*:

```
if( nextToABeeper && frontIsClear )
        move();
```

Por ejemplo, para *or*:

```
if( nextToABeeper || frontIsBlocked )
        turnoff();
```

Elaborado por:

Edgar Alfredo Duéñez Guzmán

Marte Alejandro Ramírez Ortegón