

Manual de Usuario

Juan Francisco Cabrera Sánchez Carlos Gallardo Polanco
<https://github.com/gpcarlos95/Electrical-Circuits-Language>

1 de febrero de 2018

1 Introducción

En primer lugar, se describen las palabras reservadas del lenguaje, que son las siguientes:

```
switch  button  lamp  sensor  bell  fuse  relay  junction
minute  plug    lock  regulator  movDetector  R  S  G
```

Todas estas palabras reservadas, a excepción de R, S y G, pueden además, ir acompañadas de caracteres numéricos. Por tanto, en este lenguaje, se definen las instrucciones de la siguiente forma:

```
PalabraReservada (PalabraReservada, PalabraReservada, ...)
```

Cabe destacar que, a diferencia de muchos lenguajes, aquí no se utiliza el ';' como separador de instrucciones, sino que, cada instrucción está separada por espacios en blanco o por saltos de línea indistintamente.

Como posiblemente conozca, en los circuitos eléctricos, existen diferencias entre el hecho de conectar varios componentes en serie o conectarlos en paralelo, como es el caso de las diferencias en la tensión o en la corriente.

Por ello, se ha buscado, que resulte sencillo diferenciar cuándo varios componentes se conectan en serie o en paralelo. En este caso, es tan sencillo como que, dados un conjunto de componentes: $C_1, C_2, C_3, C_4, \dots, C_n$, si todos están conectados al mismo componente por la izquierda por una parte, y al mismo componente por la derecha por otra. Análogamente, se puede describir fácilmente cuándo un conjunto de componentes está conectado en serie. Dado un conjunto de componentes: $C_1, C_2, C_3, C_4, \dots, C_n$, diremos que están conectados en serie siempre que el componente que esté a la izquierda de C_n sea C_{n-1} y el componente que esté a la derecha de C_{n-1} sea C_n . La mejor forma de comprobar esto es con un ejemplo:

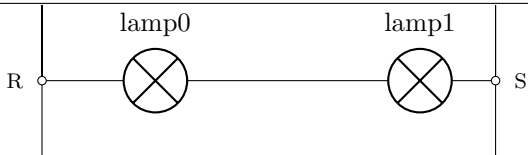
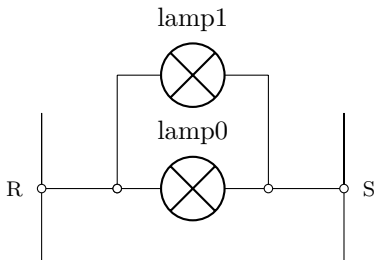
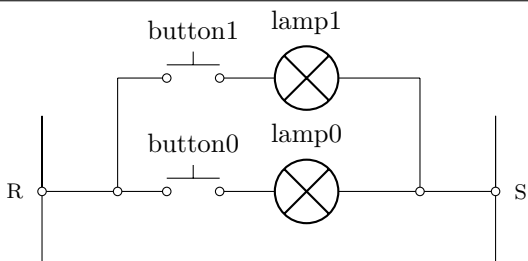
- En serie:

```
lamp1(R, button1)
button1(lamp1, fuse2)
fuse2(button1, switch3)
switch3(fuse2, bell0)
```

- En paralelo:

```
switch0 (R,S)
lamp1 (R,S)
button1 (R,S)
```

Por tanto, hay 3 posibilidades, que el circuito sea un conjunto de componentes conectados en serie, que el circuito sea un conjunto de componentes conectados en paralelo o la más común, que el circuito sea una combinación de las dos anteriores. Esto se puede observar en el siguiente cuadro:

Código	Output
<pre>lamp0 (R, lamp1) lamp1 (lamp0, S)</pre>	
<pre>lamp0 (R, S) lamp1 (R, S)</pre>	
<pre>button0 (R, lamp0) lamp0 (button0, S) button1 (R, lamp1) lamp1 (button1, S)</pre>	

Cuadro 1: Resumen de la sintaxis y la salida que produce.

Sin embargo, cuando vaya a realizar circuitos con una cierta complejidad se le recomienda hacer uso de **junction**. Este elemento permite reducir la ambigüedad al crear circuitos eléctricos. Su uso resulta muy sencillo. Tomando como ejemplo el 3er circuito del cuadro anterior, las circunferencias que observa entre las tomas R y S y los elementos, son los puntos en los que se divide(o se une) el potencial eléctrico. Esto permite que el diseño de circuitos complejos sea lo más claro posible, sin posibilidad alguna de ambigüedad. Veamos un ejemplo que ilustre el funcionamiento de este elemento.

Código	Output
<pre> junction1(R,button1) button1(junction1,lamp1) lamp1(button1,button2) button2(lamp1,junction6) junction2(junction1,junction3) junction3(junction2,lamp2) lamp2(junction3,junction5) junction4(junction2,lamp3) lamp4(junction3,junction4) lamp3(junction4,junction5) junction5(lamp2,junction6) junction6(button2,S) </pre>	

Cuadro 2: Uso del junction

Se deben tener las siguientes consideraciones a la hora de programar: no se permite la redefinición de componentes, es decir, si se intenta definir dos veces el mismo componente, habrá un mensaje de error indicando qué componente está repetido, considerándose el circuito inválido por este motivo. Recuerde además, que el circuito debe quedar cerrado, si no lo está, se considerará inválido y se notificará con el mensaje de error correspondiente.

Y, la más importante, a la hora de definir los elementos, todos tendrán como mínimo dos conexiones, ya que deben estar conectados a las tomas R y S. Puede estar conectado de forma directa o a través de otro elemento, pero el primer elemento al que se conecta debe ser R o a un elemento que conduzca a R y lo mismo ocurre con el segundo elemento, solo que esta vez, en lugar de con R es con S. Los siguientes elementos (si los tiene) a los que se conecta no deben seguir ningún criterio específico sobre el orden, a excepción del enchufe(plug) cuyo tercer conector sólo puede ir a tierra(G). Puede ver de forma más clara en el cuadro 3 los conectores que puede definir, junto con su correspondiente número de conectores.

Nombre	Conectores
Interrupor (switch)	2 o 3
Pulsador (button)	2
Lámpara (lamp)	2
Sensor (sensor)	2
Timbre (bell)	2
Fusible (fuse)	2
Relé (relay)	4-18 (número par)
Minutero (minute)	4-18 (número par)
Enchufe (plug)	2 o 3 (uno es tierra)
Cerradura eléctrica (lock)	2
Regulador (regulator)	6
Detector de movimiento (movDetector)	6
Punto de unión (junction)	2

Cuadro 3: Lista de componentes disponibles con su correspondiente número de conectores.

Si lo desea, puede añadir comentarios a su código haciendo uso de `// ...` y `/* ... */`

2 Compilación y ejecución

A partir de aquí se le explicará cómo compilar su circuito, y se le dan dos opciones:

- Usando el makefile:

Para compilarlo todo:
`make`

Para compilar el léxico:
`make flex`

Para compilar la semántica:
`make bison`

Para compilar, enlazar y generar un ejecutable:
`make analyzer`

Para ejecutar el analizador:
`make exe FILE="NombreDelCircuito.circuit"`

- Sin usar el makefile:

Para compilar el léxico:
`flex lexicon.l`

Para compilar la semántica:
`bison -d syntactic.y`

Para compilar, enlazar y generar un ejecutable:
`g++ lex.yy.c syntactic.tab.c -o analyzer -lfl -lm`

Para ejecutar el analizador:
`cat NombreDelCircuito.circuit | ./analyzer`