Capítulo 15

Manejo de Archivos

15.1. Las primitivas

Como siempre, vamos a clasificar las primitivas en función de su objetivo:

15.1.1. Navegación por el sistema de archivos

Al iniciar xLogo, la ruta de trabajo será el directorio personal (`/home/nombre/` en Linux y Mac) o el directorio raíz `C:\\` en Windows. Para desplazarnos por el disco duro y ejecutar programas externos, disponemos de:

Primitivas	Argumentos	Uso
`catalogo`, `cat`	`no`	Lista el contenido del directorio actual. (Equivalente al comando `ls` de Linux, `dir` de DOS)
`pondirectorio`, `pondir`	`lista`	Especifica el directorio actual. La ruta debe ser absoluta. El directorio debe especificarse dentro de una lista, y la ruta no debe contener espacios.
`cambiadirectorio`, `cd`	`palabra o lista`	Cambia el directorio de trabajo desde el directorio actual (ruta relativa). Puede utilizarse `..` para referirse a la ruta del directorio superior.
`directorio`, `dir`	`no`	Da el directorio actual.

Supongamos la siguiente estructura de directorios en el disco duro de un alumno:

```
/home/alumno
   --> /Documentos
         --> /Clase
         --> /xLogo
              --> /Programas
```

```
            --> /Capturas
--> /Escritorio
--> /Imagenes
```

Al iniciar xLogo, la ruta de trabajo será `/home/alumno/`. Para llegar al directorio `Programas` podemos escribir alternativamente:

```
pondirectorio "/home/alumno/Documentos/xLogo/Programas
cambiadirectorio "Documentos/xLogo/Programas
```

Una vez en él, podemos ir hacia atrás de forma absoluta o relativa:

```
pondirectorio "/home/alumno/Documentos/xLogo
cambiadirectorio "..
```

y listar su contenido, que se mostrará en el Histórico de Comandos separando los directorios de los archivos:

```
catalogo
--> Directorio(s):
    Programas Capturas
    Archivo(s):
    manual.pdf prueba.lgo tortuga.png transfo.png xlogo.jar
```

Hay dos posibilidades de que esto no sea exactamente así:

- cuando se abre un archivo `.lgo` desde el Menú Archivo, el directorio donde se encontraba éste queda establecido como directorio de trabajo. Por ejemplo, si abrimos el fichero:

  ```
  /home/alumno/Documentos/xLogo/rosa.lgo
  ```

  el directorio de trabajo será

  ```
  /home/alumno/Documentos/xLogo
  ```

- Si no es la primera vez que se trabaja con xLogo. Al terminar una "sesión", se guarda la última ruta de trabajo y se mantiene para ejecuciones posteriores.

Todo lo anterior afecta al guardado en disco de procedimientos o la carga desde disco duro de procedimientos e imágenes, NO a las capturas de pantalla que pueden hacerse desde el Menú Archivo $\longrightarrow$ Capturar la imagen.

15.1.2. Carga y guardado de procedimientos

Finalizando el capítulo 5 te enseñamos a guardar en el disco duro usando las opciones de Menú:

- Menú Archivo ⟶ Guardar como ....
- Menú Archivo ⟶ Guardar

que difieren en que uno te permite asignar un nombre, y el otro sobreescribe el archivo abierto (si lo hay). Podemos también abrir procedimientos con:

Menú Archivo ⟶ Abrir

y disponemos de primitivas con las que conseguir lo mismo:

Primitivas	Argumentos	Uso
`carga`	`palabra`	Abre y lee el archivo indicado por `palabra`.
`guarda`	`palabra`, `lista`	Guarda en el archivo indicado por `palabra` los procedimientos especificados en `lista`, en el directorio actual. (Ver ejemplo)
`guardatodo`	`palabra`	Guarda en el archivo indicado por la `palabra` todos los procedimientos definidos, en el directorio actual. (Ver ejemplo)

Ejemplos:

- `carga "prueba.lgo` carga un archivo llamado `prueba.lgo` del directorio actual, SIN ABRIR el Editor de Comandos.

  Difiere de la opción `Abrir` del Menú Archivo en que este sí abre el Editor de Comandos.
- `guarda "trabajo.lgo [proc1 proc2 proc3]` guarda en el directorio actual un archivo llamado `trabajo.lgo` que contiene los procedimientos `proc1`, `proc2` y `proc3`.
- `guardatodo "trabajo.lgo` guarda en el directorio actual un archivo llamado `trabajo.lgo` que contiene la totalidad de los procedimientos actualmente definidos.

  Es equivalente a usar la opción `Guardar` del Menú Archivo.

En ambos casos, si no se indica la extensión `.lgo`, será añadida. La palabra especifica una ruta relativa a partir del directorio corriente. No funciona colocar una ruta absoluta.

Para borrar todos los procedimientos definidos y cargar el archivo `trabajo.lgo`, debes usar:

```
borratodo carga "trabajo.lgo
```

La palabra especifica una ruta relativa a partir del directorio corriente. No funciona colocar una ruta absoluta, es decir:

```
borratodo carga "/home/alumno/xLogo/trabajo.lgo
```

no producirá ningún efecto.

15.1.3. Modificando archivos

Primitivas	Argumentos	Uso
abreflujo	número nombre	Para poder leer o escribir en un fichero, es necesario crear un flujo hacia él. El argumento nombre debe ser su nombre, que se refiere al directorio de trabajo. El argumento n es el número que identifica a ese flujo.
cierraflujo	número	Cierra el flujo n.
listaflujos	lista	Carga una lista con los flujos abiertos indicando su identificador
leelineaflujo	número	Abre el flujo cuyo identificador es n, y lee una línea del fichero
leecarflujo	número	Abre el flujo cuyo identificador es n, después lee un caracter del fichero. Esta primitiva devuelve el número correspondiente al caracter unicode (como leecar - sec. 12.2)
escribelineaflujo	número lista	Escribe la línea de texto indicada en lista al principio del fichero indicado por el flujo n. Atención: la escritura no se hace efectiva hasta que se ciera el fichero con cierraflujo.
agregalineaflujo	número lista	Escribe la línea de texto indicada en lista al final del fichero indicado por el flujo n. Atención: la escritura no se hace efectiva hasta que se ciera el fichero con cierraflujo.
finflujo?	número	Devuelve cierto si se ha llegado al final del fichero, y falso en caso contrario.

Ejemplo:

El objetivo es crear el fichero ejemplo en el directorio personal: /home/tu_nombre, en Linux, C:\, en Windows que contiene:

```
ABCDEFGHIJKLMÑNOPQRSTUVWXYZ
abcdefghijklmñnopqrstuvwxyz
0123456789
```

Para ello:

```
# abre un flujo hacia el fichero indicado
# identificara el flujo con el numero 2
pondirectorio "/home/tu_nombre
abreflujo 2 "ejemplo
# escribe las lineas que quiero
escribelineaflujo 2 [ABCDEFGHIJKLMNÑOPQRSTUVWXYZ]
escribelineaflujo 2 [abcdefghijklmnñopqrstuvwxyz]
escribelineaflujo 2 [0123456789]
# cerramos el flujo para acabar la escritura
cierraflujo 2
```

Ahora, comprobamos que está bien escrito:

```
# abre un flujo hacia el fichero indicado
# identificara el flujo con el numero 0
pondirectorio "/home/tu_nombre
abreflujo 0 "ejemplo
# lee las lineas del fichero consecutivamente
escribe leelineaflujo 0
escribe leelineaflujo 0
escribe leelineaflujo 0
# cerramos el flujo
cierraflujo 0
```

Si queremos que nuestro fichero termine con la línea `Formidable!`:

```
pondirectorio "c:\\
abreflujo 1 "ejemplo
agregalineaflujo 1 [Formidable!]
cierraflujo 1
```

15.2. Ejecutando programas externos

Si quisiéramos ejecutar un program externo a xLogo, disponemos de la primitiva `comandoexterno`. Su argumento debe ser una lista de sub–listas, que contienen en este orden:

- El comando que lanza el programa
- Opcionalmente, las opciones del mismo

Por ejemplo:

```
comandoexterno [ [gedit] ]
```

Ejecuta el program GEDIT (en Linux) sin opciones.

```
comandoexterno [ [notepad] ]
```

Ejecuta el program BLOCK DE BOTAS (en Windows) sin opciones.

```
comandoexterno [ [gedit] [/home/xlogo/ejemplo.txt] ]
```

Abre el archivo `ejemplo.txt` con GEDIT (en Linux).

```
comandoexterno [ [notepad] [c:\ejemplo.txt]]
```

Abre el archivo `ejemplo.txt` con el BLOCK DE NOTAS (en Windows).

Esta sintaxis tan "especial" permite llamar al sistema con los espacios en blanco adecuados para que no haya errores. Observa, también, que la ruta a los archivos debe ser **absoluta**.

15.2.1. Ejemplo: La tortuga ... ¡Habla!

Si disponemos de un sintetizador de voz en nuestro sistema operativo, algo habitual si nuestro sistema dispone de Herramientas de Accesibilidad, podemos oír hablar a la tortuga.

En este caso vamos a desarrollar un ejemplo sobre `Festival`, cuyo verdadero nombre es **Festival Speech Synthesis System**, un sistema muy completo para convertir texto en sonido y viceversa. Podemos encontrar más información en:

`http://www.cstr.ed.ac.uk/projects/festival`

Vamos a crear un archivo de texto con las órdenes para Festival, de momento un sencillo saludo y la orden de salida, y después llamaremos al sistema con `comandoexterno`. El problema no está en crear la secuencia de órdenes, sino en escribir la ruta completa al archivo.

Usaremos `directorio` y la forma general de `lista`:

```
para habla
  abreflujo 1 "habla                              # Abrimos el archivo de ordenes
  escribelineaflujo 1 [(SayText "Hola, amigo.]  # Frases que vamos a escuchar
  escribelineaflujo 1 [Como estas")]
  escribelineaflujo 1 [(quit)]                    # Salimos de Festival
  cierraflujo 1                                   # Cerramos el archivo
#
# Creamos las ordenes como lista de listas y con la ruta completa al archivo
#
  comandoexterno lista [festival] (lista palabra directorio "/habla)
fin
```

Una segunda posibilidad consiste en utilizar las opciones de `Festival`. En el archivo solo introducimos las frases que queremos escuchar y pasamos la opción `--tts`:

```
para habla2
  abreflujo 1 "habla2                          # Abrimos el archivo de ordenes
  escribelineaflujo 1 [Hola, amigo.]           # Frases que vamos a escuchar
  escribelineaflujo 1 [Como estas.]
  cierraflujo 1                                # Cerramos el archivo
  comandoexterno (lista [festival] [--tts] (lista palabra directorio "/habla2))
fin
```

Como ves, mucho más fácil.

15.3. Obtención aproximada de π (2)

Un resultado conocido de teoría de los números pone de manifiesto que la probabilidad que dos números tomados aleatoriamente sean primos entre ellos es de $\frac{6}{\pi^2} \simeq 0{,}6079$. Para intentar encontrar este resultado, vamos a:

- Tomar dos números al azar entre 0 y 1 000 000.
- Calcular su m.c.d.
- Si su m.c.d. vale 1, añadir 1 a una variable contador.
- Repetir 1000 veces
- La frecuencia de los pares de números primos entre ellos se obtendrá dividiendo la variable contador por 1000 (el número de pruebas)

15.3.1. Noción de m.c.d. (máximo común divisor)

Dados dos números enteros, el máximo común divisor define al mayor divisor común de ambos. Por ejemplo:

- 42 y 28 tienen como m.c.d. 14, ya que es el número más grande por el que es posible dividir a la vez 28 y 42
- el m.c.d. de 25 y 55 es 5
- 42 y 23 tienen m.c.d. igual a 1

Cuando dos números tienen m.c.d. 1, se dice que son primos entre sí. Así en el ejemplo anterior, 42 y 23 son primos entre sí. Eso significa que no tienen ningún divisor común excepto 1 (¡por supuesto, hablamos de división entera!).

15.3.2. Algoritmo de Euclides

Para determinar del m.c.d. de dos números, se puede utilizar un método llamado **algoritmo de Euclides**: (Aquí no se demostrará la validez de este algoritmo, se admite que funciona).

El mecanismo de este método es: "dados dos números naturales `a` y `b`, analizamos si `b` es nulo.:

- En caso afirmativo, entonces el m.c.d. es igual a `a`.
- Si no, se calcula `r`, el resto de la división de `a` entre `b`.
- Se sustituyen `a` por `b` y `b` por `r`, y se reinicia el método.

Calculemos por ejemplo, el m.c.d. de 2160 y 888 por este algoritmo con las siguientes etapas:

a	b	r
2160	888	384
888	384	120
384	120	24
120	24	0
24	0	

El m.c.d. de 2160 y 888 es, por tanto, 24. 24 es el mayor entero que divide simultáneamente a los dos números. De hecho, `2160 = 24 * 90` y `888 = 24 * 37`. El m.c.d. es, por tanto, el último resto no nulo.

15.3.3. Calcular un m.c.d. en xLogo

Un pequeño algoritmo recursivo permite calcular el m.c.d. de dos números, `:a` y `:b`.

```
para mcd :a :b
   si (resto :a :b) = 0
      [devuelve :b]
      [devuelve mcd :b resto :a :b]
fin

escribe mcd 2160 888
```

proporciona como resultado `24`

Nota: Nos vemos obligados a poner paréntesis en `resto :a :b`, si no el intérprete intentará evaluar `b = 0`. Para evitar este problema de paréntesis, podemos escribir: `si 0 = resto :a :b`

15.3.4. Avanzando con el programa

Tras la introducción matemática, recordemos nuestro objetivo:

- Tomar dos números al azar entre 0 y 1 000 000.
- Calcular su m.c.d.
- Si su m.c.d. vale 1, añadir 1 a una variable contador.
- Repetir 1000 veces
- La frecuencia de los pares de números primos entre ellos se obtendrá dividiendo la variable contador por 1000 (el número de pruebas)

```
para prueba
# Inicializamos la variable contador a 0
  haz "contador 0
  repite 1000
    [ si (mcd azar 1000000 azar 1000000) = 1
       [haz "contador :contador+1] ]
  escribe [Frecuencia:]
  escribe :contador/1000
fin
```

Comprobamos la validez del método:

```
prueba
0.609
prueba
0.626
prueba
0.597
```

y vemos que se obtienen valores próximos al valor teórico de 0,6097. Lo que es notable es que esta frecuencia es un valor aproximado de $\frac{6}{\pi^2} \simeq 0,6079$.

Si tenemos en cuenta `f`, la frecuencia encontrada, se tiene, entonces:

$$f \simeq \frac{6}{\pi^2}$$

Despejando:

$$\pi^2 \simeq \frac{6}{f} \quad \text{y así:} \quad \pi \simeq \sqrt{\frac{6}{f}}$$

Añadimos esta aproximación al programa, y transformamos el final del procedimiento `prueba`:

```
para prueba
# Inicializamos la variable contador a 0
  haz "contador 0
  repite 1000
    [ si (mcd azar 1000000 azar 1000000) = 1
       [ haz "contador :contador+1 ] ]
# Tras calcular la frecuencia
  haz "f :contador/1000
# Mostramos el valor aproximado de pi
  escribe frase [Aproximacion de pi:] raizcuadrada (6/:f) fin
fin
```

que proporciona:

```
prueba
Aproximacion de pi: 3.164916190172819
prueba
Aproximacion de pi: 3.1675613357997525
prueba
Aproximacion de pi: 3.1008683647302115
```

Por último, modifiquemos el programa de modo que cuando lo lancemos, podamos indicar cuántas pruebas con números aleatorios deseamos.

```
para prueba :repeticiones
# Inicializamos la variable contador a 0
  haz "contador 0
  repite :repeticiones
    [ si (mcd azar 1000000 azar 1000000) = 1
       [ haz "contador :contador+1 ] ]
# Tras calcular la frecuencia
  haz "f :contador/:repeticiones
# Mostramos el valor aproximado de pi
  escribe frase [Aproximacion de pi:] raizcuadrada (6/:f)
fin
```

Probamos con 10000 repeticiones, y obtenemos en las tres primeras tentativas:

```
prueba 10000
Aproximacion de pi: 3.1300987144363774
prueba 10000
Aproximacion de pi: 3.1517891481565017
prueba 10000
Aproximacion de pi: 3.1416626832299914
```

No está mal, ¿verdad?

15.4. Compliquemos un poco más: π que genera π...

¿Qué es un número aleatorio? ¿Es que un número tomado aleatoriamente entre 1 y 1.000.000 es un número realmente aleatorio?

Deberías darte cuenta rápidamente que nuestro modelo no hace más que aproximarse al modelo ideal. Bien, es precisamente sobre el modo de generar el número aleatorio sobre el que vamos a efectuar algunos cambios .... No vamos utilizar más la primitiva `azar`, sino que utilizaremos la secuencia de los decimales de π.

Me explico: los decimales de π siempre han intrigado a los matemáticos por su falta de regularidad, las cifras de 0 a 9 parecen aparecer en cantidades aproximadamente iguales y de manera aleatoria. No se pueden predecir los decimales siguientes basándonos en los anteriores.

Vamos a ver a continuación como generar un número aleatorio con ayuda de los decimales de π. En primer lugar, debes obtener los primeros decimales de π (por ejemplo un millón). Existen dos programas que los calculan bastante bien. Aconsejamos `PiFast` para Windows y `Schnell-Pi` para Linux.

Puedes acceder a Internet para conseguir un fichero de texto:

`http://3.14159265358979323846264338327950288419716939937510582097494459 2.com`

También en Internet, en la *web* de xLogo:

`http://downloads.tuxfamily.org/xlogo/common/millionpi.txt`

Para crear los números aleatorios, agrupamos de 8 en 8 cifras la serie de decimales de π. Es decir, el fichero empieza así:

$$\underbrace{3{,}1415926}_{1^{\text{er}}\ \text{numero}}\ \underbrace{53589793}_{2^{\text{o}}\ \text{numero}}\ \underbrace{23846264}_{3^{\text{er}}\ \text{numero}}\ \underbrace{33832795}_{\dots}\ 0288419716939\dots$$

Retiro la coma "," del `3.14...` que podría equivocarnos al extraer los decimales. Bien, todo está preparado, creamos un nuevo procedimiento llamado `azarpi` y modificamos ligeramente el procedimiento `prueba` para llamar al procedimiento `azarpi`:

```
para azarpi :n
  hazlocal "numero "
  repite :n [
# Si no hay ningun caracter en la linea
  si 0=cuenta :linea
   [haz "linea primero leelineaflujo 1]
# Asignamos a la variable :caracter el valor de primer caracter de la linea
  haz "caracter primero :linea
# despues eliminamos el primer caracter de la linea
  haz "linea menosprimero :linea
  haz "numero palabra :numero :caracter ]
  devuelve :numero
fin
```

El resultado es:

```
prueba 10
Aproximacion de pi: 3.4641016151377544
prueba 100
Aproximacion de pi: 3.1108550841912757
prueba 1000
Aproximacion de pi: 3.081180112566604
prueba 10000
Aproximacion de pi: 3.1403714651066386
```

Encontramos pues una aproximación del número π ¡con ayuda de sus propios decimales!