Índice de
Subrutinas y funciones predefinidas de SL

Funciones matemáticas

- abs()
- arctan()
- cos()
- exp()
- int()
- log()
- sin()
- sqrt()
- tan()

Entrada/salida

- beep()
- cls()
- imprimir()
- leer()
- eof()
- get_color()
- get_curpos()
- get_ifs()
- get_ofs()
- get_scrsize
- readkey()
- set_color()
- set_curpos()
- set_ifs()
- set_ofs()
- set_stdin()
- set_stdout()

Funciones de cadenas de caracteres

- ascii()
- lower()
- ord()
- pos()
- strdup()
- strlen()
- substr()
- upper()

Transormación de datos

- str()
- val()

Arreglos

- alen()
- dim()

Otros

- dec()
- ifval()
- inc()
- intercambiar()
- max()
- min()
- paramval()
- pcount()
- random()
- runcmd()
- sec()
- swap()
- terminar()

jsegovia@cnc.una.py

abs()

Retorna el valor absoluto de un número.

Sintaxis

```
sub abs (n : numerico) retorna numerico
```

Ejemplo

Suponga que desea imprimir la distancia en kilómetros que existe entre dos ciudades, localizadas ambas sobre la misma ruta. El usuario ingresa la distancia de ambas ciudades con relación a un punto inicial (por ejemplo Asunción) y el programa imprime la distancia entre ambas.

```
var
   c1, c2 : numerico
inicio
   imprimir ("\nIngrese la distancia de la ciudad A desde Asunción:")
   leer (c1)
   imprimir (Ingrese la distancia de la ciudad B desde Asunción:")
   leer (c2)
   imprimir ("Entre A y B existen ", abs(c1-c2),
             " kilómetro(s)")
fin
```

Usando la función `abs()` puede asegurarse que nunca verá un valor negativo como resultado de la resta, que bien podría dar un resultado negativo.

beep()

Emite un pitido a cierta frecuencia y la ejecución se suspende durante el tiempo que se indica en el segundo parámetro (expresado en milisegundos).

Ambos parámetros pueden omitirse, en cuyo caso se utiliza automáticamente los valores (500, 100), es decir, se emite un pitido y el programa se detiene durante una décima de segundo.

Si la salida actual es un archivo, el pitido no se emite, aunque se hace la pausa solicitada.

Nótese que `beep()` puede ser utilizado para introducir una pausa en la ejecución, sin emitir pitido propiamente. Por ejemplo `beep(0, 500)` hará una pausa de al menos medio segundo.

Sintaxis

```
sub beep (frecuencia, duración_miliseg : numerico)
```

Ejemplo

```
var
   t0, t1 : numerico
inicio
   imprimir (“\nIniciamos con un pitido.”)
   beep()
   imprimir (“\nAhora haremos una pausa de 3 segundos\n”)
   t0 = sec()
   beep (0, 3000)
   t1 = sec()
   beep()
   imprimir (“Pausa finalizada. Duró ”, t1-t0, “ segundos”)
fin
```

---

alen()

Retorna la cantidad de elementos que componen un arreglo. La función opera por igual sobre arreglos abiertos y no abiertos.

Sintaxis

```
sub alen (nombre_var) retorna numerico
```

Ejemplo

El ejemplo que sigue imprimirá la cantidad de elementos de que consta A, que depende de qué valor se ingrese para la variable n. Además imprimirá:

```
La matriz M tiene 3 filas
Todas las filas tienen 4 columnas.
```

```
var
   A : vector [10] numerico
   M : matriz [*, *] cadena
   n = 0
inicio
   imprimir (“\nIngrese tamaño del vector A:”)
   leer (n)

   dim (A, n)
   imprimir (“\nLa cantidad de elementos de A es “, alen (A))

   dim (M, 3, 4)
   imprimir (“\nLa matriz M tiene “, alen (M), “ filas.”,
             “\nTodas las filas tienen “, alen (M [1]), “ columnas.”)
fin
```

---

arctan()

Calcula el arco tangente (o la inversa de la tangente) de un ángulo, expresado

en radianes.

Si tan(w) es z, arctan(z) es w.

Sintaxis

```
sub arctan (a : numerico) retorna numerico
```

ascii()

Retorna el carácter que se encuentra en la tabla ASCII en la posición dada.

Sintaxis

```
sub ascii (n : numerico) retorna cadena
```

Ejemplo

El programa que sigue imprime todos los dígitos del “0” al “9” y sus posiciones en la tabla ASCII. Para averiguar la posición del “0”, es decir, la posición inicial de la serie, se usa la función `ord()`.

```
var
   pos_0 = ord (“0”)
   k     = 0
inicio
   desde k=pos_0 hasta pos_0 + 9 {
      imprimir (“\nCarácter “, ascii (k), “. Posición “, k)
   }
fin
```

cls()

Limpia la pantalla.

Sintaxis

```
sub cls()
```

La pantalla se limpia solo si el dispositivo de salida al momento de ejecutarse `cls()` es la pantalla. Si la salida está redireccionada vía `set_stdout()`a un archivo, `cls()` no tiene ningún efecto.

cos()

Calcula el coseno de un ángulo expresado en radianes.

Sintaxis

```
sub cos (a : numerico) retorna numerico
```

dec()

Decrementa el valor de la variable que se le pasa como parámetro. En su forma básica, es equivalente a hacer c=c-1, donde c es una variable numérica.

Sintaxis

```
sub dec (ref n : numerico; decr : numerico) retorna numérico
```

El segundo parámetro es opcional; si se omite, el compilador de SL asume que es el valor 1. `dec()` realiza la operación contraria a `inc()`.

Se retorna el nuevo valor de la variable luego del decremento; este valor es habitualmente ignorado.

Ejemplo

Vea `inc()`.

dim()

Asigna memoria para los elementos de una matriz o vector que haya sido declarado como “abierto”, es decir, con asterisco (*) como tamaño.

Use `dim()` y arreglos “abiertos” cuando la cantidad de elementos que se requiere depende de un valor que el usuario ingresa, o de un valor que se calcula. En cambio, si la cantidad de elementos ya se conoce al momento de preparar el programa, se puede “dimensionar” ya el arreglo con tal tamaño.

Sintaxis

```
sub dim (nombre_var, tam_dim1, tam_dim2, ...)
```

Ejemplo

El siguiente ejemplo asigna n elementos al vector A, donde el valor de n es proveído por el usuario, así como los valores que serán almacenados en cada posición.

También inicializa la matriz M con 3 filas y 4 columnas. Muestra también cómo se pueden leer los elementos de un arreglo con una sola llamada a `leer()`; en este caso se tomarán 12 valores.

Finalmente, la matriz Z es peculiar en cuanto que una de sus dimensiones es "abierta", pero la segunda ya tiene un tamaño definido (5 en este caso). Nótese que en estos casos `dim()` solo recibe tamaños para las dimensiones "abiertas"; las demás son automáticamente completadas por el compilador de SL.

```
var
   A : vector [*] numerico
   M : matriz [*, *] cadena
   Z : matriz [*, 5] numerico
   n = 0
inicio
   imprimir ("\nIngrese tamaño del vector A:")
   leer (n)
   dim (A, n)
   desde k=1 hasta n {
      leer (A[k])
   }

   dim (M, 3, 4)
   leer (M)

   dim (Z, n*2)
   desde k=1 hasta alen(Z) {
      /*
       * Por cada iteración, se leerán 5 valores a la vez, ya que
       * cada Z [k] contiene tal cantidad de elementos.
       */
      leer (Z [k])
   }
fin
```

---

eof()

Retorna verdadero cuando ya no existen datos que puedan ser leídos desde un archivo.

Sintaxis

```
sub eof() retorna logico
```

Ejemplo

El siguiente código imprime cada línea de un archivo llamado datos.txt, mostrando el número de línea correspondiente.

```
var
   linea = ""
   num_linea = 0
inicio
   set_stdin ("datos.txt")
   set_ifs ("\n")
   leer (linea)
   mientras ( not eof() ) {
      inc (num_linea)
      imprimir ("\n", num_linea, ": ", linea)
      leer (linea)
   }
   imprimir ("\nFueron leídas ", num_linea, " líneas.")
fin
```

exp()

Retorna e elevado a la n, donde e es la base de los logaritmos naturales o neperianos (2.71828182...).

Sintaxis

```
sub exp (n : numerico) retorna numerico
```

Ejemplo

```
var
   base_log_naturales = 0
inicio
   base_log_naturales = exp (1)
   imprimir ("\nLa base de los logaritmos naturales es ", base_log_naturales)
fin
```

get_color()

Obtiene el color del texto y del fondo vigentes. Nótese que ambos parámetros son pasados por referencia, por lo que deben ser proveídos dos nombres de variables.

Los colores están identificados por un valor entero.

Sintaxis

```
sub get_color (ref primer_plano, fondo : numerico)
```

Ejemplo

```
var
   prim_plano, fondo : numerico
```

```
inicio
   get_color (prim_plano, fondo)
   imprimir (“\nLos colores actuales son: Fondo=”, fondo,
             “ Texto=”, prim_plano)
fin
```

get_curpos()

Informa la línea y columna donde se encuentra actualmente el cursor. Nótese que los dos parámetros que necesita esta rutina son pasados por referencia, es decir, deben ser nombres de variables numéricas.

Ambos parámetros retornarán con cero si la salida actual es un archivo y no la pantalla.

Sintaxis

```
sub get_curpos (ref linea_actual, col_actual : numerico)
```

Ejemplo

```
var
   cursor_linea, cursor_col : numerico
inicio
   imprimir (“\nBuenos días.”)
   get_curpos (cursor_linea, cursor_col)
   imprimir (“\nBuenas tardes”)
   set_curpos (cursor_linea, cursor_col+1)
   imprimir (“Esto debe mostrarse al lado del saludo inicial.”)
fin
```

get_ifs()

Retorna el carácter que está vigente como separador de valores o campos en una operación de lectura vía `leer()`.

Esta función puede usarse para cambiar temporalmente el separador de campos, usando `set_ifs()`, y posteriormente restaurar su valor original.

Nota: Esta función rara vez se utiliza.

Sintaxis

```
sub get_ifs() retorna cadena
```

get_ofs()

Retorna el carácter que está vigente como separador de valores al imprimir, así como el indicador de que un arreglo no está dimensionado. El primero es un solo carácter y ocupa la primera posición en el valor retornado; el segundo inicia en la posición dos.

Sintaxis

```
sub get_ofs() retorna cadena
```

get_scrsize()

Obtiene la cantidad de líneas y columnas que tiene la pantalla. Nótese que ambos parámetros son pasados por referencia, por lo que se deben proveer dos nombres de variables. El tamaño habitual de la pantalla es 25 líneas por 80 columnas.

Sintaxis

```
sub get_scrsize (ref cant_lin, cant_cols : numerico)
```

ifval()

Dada una condición y dos expresiones e1 y e2, `ifval()` retorna el valor de e1 si la condición es verdadera y el valor de e2 si es falsa.

Si bien `ifval()` luce como una función cualquiera, es diferente en cuanto que evalúa solo una de las expresiones, es decir, implementa lo que se conoce con el nombre de "`expresión condicional`".

Las expresiones e1 y e2 deben coincidir en tipo y pueden corresponder a cualquier tipo simple (`numerico`, `cadena`, `logico`).

El uso de `ifval()` suele ayudar a hacer más compacto el código.

Sintaxis

```
sub ifval (condicion : logico
           <expresion_si_condicion_es_verdadera>
           <expresion_si_condicion_es_falsa>) retorna <valor>
```

Ejemplo

El siguiente ejemplo implementa una subrutina que retorna el nombre del día (“lun”, “mar”, ...) dado su número de día (1 a 7). Obsérvese cómo la rutina usa `ifval()` para protegerse de que se le envíe un valor fuera de rango. Sin esta validación, el programa podría interrumpirse, por estar fuera de rango el índice de un vector.

Como podrá notarse, esta misma validación puede implementarse con una sentencia condicional “si”, solo que el código con `ifval()`, en este caso, es corto e igualmente legible.

```
inicio
   imprimir (“\nEl número de día 3 es “, nombre_dia (3))
   imprimir (“\nEl número de día 9 es “, nombre_dia (9))
fin



sub nombre_dia (num_dia : numerico) retorna cadena
/*
 * Retorna el nombre del día, dado su número (1=domingo, 2=lunes, ...)
 * Si el número de día está fuera de rango, se retorna el texto
 * “MAL”
 */
var
   nom_dias : vector [*] cadena = {“dom”, “lun”, “mar”, “mie”,
                                   “jue”, “vie”, “sab”
                                  }
inicio
   retorna ifval (num_dia>0 and num_dia<8, nom_dias [num_dia], “MAL”)
fin
```

imprimir()

Muestra valores en la pantalla, o los graba en un archivo.

Sintaxis

sub imprimir (valor_1, valor_2, ...)

Los valores se muestran o graban en la posición inmediatamente disponible, sin mediar ningún carácter separador. Los valores estructurados se imprimen con cierto formato que se explica más adelante.

Para facilitar el formateo de los datos, imprimir() reconoce ciertas secuencias de caracteres como especiales (se las suele llamar “secuencias de escape”):

Secuencia	Significado
\n	Salto de línea. Cada vez que se encuentre esta secuencia, por sí misma o incrustada en otra cadena, el cursor se mueve a la siguiente línea, en la primera columna. Por lo tanto, todo lo que se imprima a continuación aparece en una línea aparte.

\t	Tabulador. Hace que el cursor salte a la posición de la siguiente tabulación. Cada columna de tabulación ocupa 8 caracteres. Puede usarse para encolumnar datos que componen una tabla.
\\	El carácter “\”.
\c	En este caso “c” simboliza cualquier carácter. Por ejemplo “\”” hará que se imprima una comilla doble.

Ejemplo 1

El programa que sigue imprime la tabla de cuadrados y cubos de los números del 1 al 10.

```
var
   n, c2, c3 : numerico
inicio
   imprimir ("\nNúmero\tCubo\tCuadrado")
   desde n=1 hasta 10 {
      imprimir ("\n", n, "\t", n*n*n, "\t", n*n)
   }
fin
```

El resultado debe lucir como sigue:

```
Número  Cubo    Cuadrado
1       1       1
2       8       4
3       27      9
4       64      16
5       125     25
6       216     36
7       343     49
8       512     64
9       729     81
10      1000    100
```

Ejemplo 2

El siguiente código carga en cada posición k del vector A el cuadrado de k. Muestra el vector con una sola llamada a imprimir().

```
var
   A : vector [50] numerico
   k : numerico
inicio
   desde k=1 hasta alen(A) {
      A [k] = k*k
   }
   imprimir (“\nResultado: ”, A)
fin
```

Resultado esperado:

```
Resultado:
1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,529
,576,625,676,729,784,841,900,961,1024,1089,1156,1225,1296,1369,1444,1521,1600,1
681,1764,1849,1936,2025,2116,2209,2304,2401,2500
```

Se puede imprimir también matrices y registros; vectores de registros y cualquier combinación de estos. Los valores siempre se separan unos de otros con una coma, o con el carácter que se haya configurado usando `set_ofs()`.

Si al imprimir un dato estructurado se encuentra un arreglo no dimensionado, se imprime por defecto **"<nodim>"**. Esto se puede cambiar con `set_ofs()`.

---

inc()

Incrementa el valor de la variable que se le pasa como parámetro. En su forma básica, es equivalente a hacer c=c+1, donde c es una variable numérica.

Sintaxis

sub inc (**ref** n : numerico; incr : numerico) **retorna** numérico

El segundo parámetro es opcional; si se omite, el compilador de SL asume que es el valor 1. El incremento puede ser positivo o negativo. `inc()` realiza la operación contraria a `dec()`.

Retorna el valor de la variable luego del incremento; este valor es habitualmente ignorado.

Usando `inc()` se puede a veces acortar el programa y hacerlo más legible, especialmente cuando se están manipulando elementos de variables estructuradas, como matrices y registros. Esto surge como consecuencia de que la variable a ser modificada se referencia una sola vez. El programa resultante será también ligeramente más rápido.

Note que es más corto y hay menos posibilidad de error al escribir

```
inc (M [mes, k*2+3])
```

en vez de

```
M [mes, k*2+3] = M [mes, k*2+3] + 1
```

Ejemplo

Vea el ejemplo que se da en ord().

---

int()

Retorna la parte entera del valor numérico que se le pasa como parámetro.

Sintaxis

```
sub int (n : numerico) retorna numérico
```

Ejemplo

Este programa lee un valor numérico positivo, con o sin decimales, y lo redondea por exceso a 2 decimales.

```
var
   n = 0
   r = 0
inicio
   imprimir ("\nIngrese un valor numérico positivo con decimales:")
   leer (n)
   /*
    * Sumamos 0.5 para que el redondeo sea por exceso.
    */
   r = int ((n*100)+0.5) / 100
   imprimir ("Valor original=", n, " Valor redondeado=", r)
fin
```

intercambiar()

Intercambia el contenido de dos variables que se pasan como parámetros. Las variables deben coincidir en tipo y pueden corresponder a valores simples, arreglos, registros o cualquier combinación de estos.

El nombre tradicional de esta rutina es `swap()`, que quiere decir precisamente intercambiar en inglés. Por ese motivo, también puede usarse `swap()` como sinónimo de `intercambiar()`.

Sintaxis

```
sub intercambiar (<variable_1>, <variable_2>)
```

Ejemplo

```
var
   a = 100
   b = 30
   M = vector [5] numerico = {10, 14, 21, 3, 1}
   N = vector [5] numerico = {1, 212, 31, 4, 90}
inicio
   imprimir ("\nAntes de intercambiar: a=", a, " b=", b)
   intercambiar (a, b)
   imprimir ("\nLuego de intercambiar: a=", a, " b=", b)

   imprimir ("\nAntes de intercambiar: M=", M, " N=", N)
   intercambiar (M, N)
   imprimir ("\nLuego de intercambiar: M=", M, " N=", N)
fin
```

leer()

Espera a que el usuario ingrese datos y pulse la tecla ENTER. Lo que se haya tipeado se asigna como nuevo valor de la variable que se haya indicado a `leer()` como parte de sus parámetros. El valor previo de la variable se pierde.

Con una sola llamada a `leer()` es posible aceptar todo tipo de valores, tanto los simples (por ejemplo, un nombre o un valor numérico) como los estructurados (arreglos y registros).

Además, `leer()` puede aceptar datos del teclado o de un archivo, indistintamente, dependiendo de cuál sea el dispositivo o archivo de entrada del momento. Consulte la función `set_stdin()` para ver cómo puede cambiarse este parámetro de ejecución.

Por defecto, cada valor a ser leído debe estar separado del siguiente por una coma. A este carácter se lo llama tradicionalmente IFS (Input Field Separator) y puede ser cambiado con la función `set_ifs()`.

Cuando se está leyendo de un archivo, es posible que se hayan leído todos los datos, o no haya suficientes datos para asignar valores a todas las variables. En tal caso, `leer()` no generará ningún error, pero la función predefinida `eof()` retornará verdadero.

El separador de líneas se considera también un separador de campos (`ASCII 13` en UNIX, `ASCII 10` seguido de `ASCII 13` en DOS/Windows).

Si se desea leer cada carácter del archivo, uno a la vez, se puede utilizar `set_ifs("")`. Consulte esta función para más detalles.

Sintaxis

```
sub leer (variable_1, variable_2, ...)
```

Ejemplo 1

El primer ejemplo muestra el uso básico de `leer()`. Simplemente se pide al usuario que ingrese dos datos, donde el primero es una cadena (nombre) y el segundo un valor numérico.

```
var
   año_nac  = 0
   nombre_completo = ""
inicio
   imprimir ("\nIngrese su nombre completo y el año de su nacimiento (aaaa):")
   leer (nombre_completo, año_nac)
   imprimir ("\nBuenos días ", nombre_completo, ".\n")
   si ( año_nac > 1985 ) {
      imprimir ("Adelante! Hay mucho camino por andar.")
```

```
   sino
      imprimir (“Descansa un poco; mucho ya has caminado.”)
   }
fin
```

Ejemplo 2

Suponga que en el ejemplo anterior se ingresase el nombre y el apellido separados por una coma. En tal caso la lectura fallaría, pues la coma es el separador por defecto y el apellido se intentaría asignar a la variable año_nac, que es un numérico, con lo que se producirá un error de ejecución.

Usaremos `set_ifs()` para que el separador de valores sea la barra de división, en vez de la coma.

```
var
   año_nac  = 0
   nombre_completo = “”
inicio
   set_ifs (“/”)
   imprimir (“\nIngrese su nombre completo y el año de su nacimiento (aaaa).”)
   imprimir (“\nUse la / para separar ambos valores”)
   leer (nombre_completo, año_nac)
   imprimir (“\nBuenos días “, nombre_completo, “.\n”)
   si ( año_nac > 1985 ) {
      imprimir (“Adelante! Hay mucho camino por andar.”)
   sino
      imprimir (“Descansa un poco; mucho ya has caminado.”)
   }
fin
```

Ejemplo 3

El siguiente ejemplo muestra el uso de leer() con arreglos. En este caso, todos los valores del vector A pueden ser ingresados en una única operación de lectura. Cada valor se puede separar del siguiente con una coma.

Los valores ingresados son mostrados en la pantalla usando también una sola llamada a `imprimir()`.

```
var
   A : vector [10] numerico
inicio
   imprimir (“\nIngrese “, alen (A), “ valores numéricos:\n”)
   leer (A)
   imprimir (“\nLos valores ingresados fueron:\n”)
   imprimir (A)
fin
```

Ejemplo 4. Lectura de registros

Supóngase que se tiene un archivo llamado “`notas.txt`” que contiene, en cada registro, la cédula, el puntaje de dos parciales y el puntaje del examen final de dicho alumno. Todas los puntajes están expresados como porcentajes.

El siguiente programa instruye a `leer()` para que lea cada registro íntegramente con una sola operación; esto se logra poniendo el nombre del registro como parámetro, en vez de cada uno de sus campos.

```
var
```

```
   r : registro {
          cedula : cadena
          p1, p2, final : numerico
       }
   prom_parc     = 0
   puntaje_final = 0
inicio
   set_stdin ("notas.txt")
   leer (r)
   /*
    * Supongamos que el último registro, no procesable ya, tiene un asterisco
    * en el campo cédula.
    */
   mientras ( r.cedula <> "*" ) {
      prom_parc = (r.p1 + r.p2)/2
      /*
       * Se asume que los parciales tienen una influencia del 30% sobre el
       * puntaje final.
       */
      puntaje_final = (0.3*prom_parc) + (0.7*r.final)
      imprimir ("\nPromedio de parciales=", prom_parc,
                " Puntaje final=", puntaje_final)
      leer (r)
   }
fin
```

log()

Calcula el logaritmo base 10 de un número n mayor a cero.

Si el número n es menor o igual a cero, se produce un error de ejecución y el programa terminará.

Sintaxis

```
sub log (n : numerico) retorna numerico
```

lower()

Dada una cadena como parámetro, retorna una nueva cadena que consta de los mismos caracteres del parámetro, pero con aquellos que sean alfabéticos convertidos a minúsculas.

Vea también upper().

Sintaxis

```
sub lower (c : cadena) retorna cadena
```

max()

Dados dos valores simples, retorna una copia del mayor de ellos. Los valores pueden ser numéricos, cadenas o lógicos y ambos deben coincidir en tipo.

La cadena vacía “” se considera que es menor a cualquier cadena no vacía.

Vea también la función min().

Sintaxis

```
sub max (valor_1, valor_2) retorna <valor_mayor>
```

Ejemplo

El siguiente ejemplo imprime el nombre de la persona que ocuparía la última posición en una lista, si esta fuese ordenada ascendentemente.

```
var
   nombres : vector [*] cadena = {“Pablo”, “Maria”, “Ana”, “Marco”}
   ult = “”
   k : numerico

inicio
   desde k=1 hasta alen(nombres) {
      ult = max (ult, nombres [k])
   }
   imprimir (“\nLa última persona sería “, ult)
fin
```

min()

Dados dos valores simples, retorna una copia del menor de ellos. Los valores pueden ser numéricos, cadenas o lógicos y ambos deben coincidir en tipo.

Vea también la función max().

Sintaxis

```
sub min (valor_1, valor_2) retorna <menor_valor>
```

Ejemplo

Dados tres valores numéricos, imprimir el menor de ellos, si es que existe tal menor valor.

```
var
   a, b, c : numerico
inicio
   imprimir (“\nIngrese tres valores numéricos:”)
   leer (a, b, c)
   si ( a == b && b == c ) {
```

```
      imprimir (“\nLos tres valores son iguales; no existe un valor menor”)
   sino
      imprimir (“\nEl menor valor es “, min (min (a, b), c))
   }
fin
```

ord()

Retorna la posición que ocupa un determinado carácter en la tabla ASCII.

Sintaxis

```
sub ord (c : cadena) retorna numerico
```

Si c contiene más de un carácter, solo se usa el primero de ellos.

Ejemplo

Este programa calcula cuántas veces aparece cada letra mayúscula del alfabeto inglés (A a Z, sin Ñ) en una cadena que el usuario ingresa.

Se utiliza el vector F para almacenar la frecuencia de aparición de cada carácter. La primera posición corresponde a la letra “A”, la segunda a la “B” y así sucesivamente. Como la letra “A” se encuentra en la posición 65 en la tabla ASCII, la “B” en la posición 66 y así sucesivamente, se utiliza la función `ord()` para asociar 1 a la “A”, 2 a la “B”, etc, a través de una sencilla fórmula.

```
var
   pos_A = ord (“A”)
   k     = 0
   z     = “”
   c     = “”
   F     :  vector [26] numerico
inicio
   leer (z)
   F = {0, ...}
   desde k=1 hasta strlen (z) {
      c = upper (z [k])
      /*
       * Como la “A” está en la posición 65 en la tabla ASCII,
       * convertimos 65 en 1, 66 en 2, etc. usando la expresión
       *    ord (c) – pos_A + 1
       */
      si ( c >= “A” && c <= “Z” ) {
         inc (F [ord (c) – pos_A + 1])
      }
   }

   desde k=1 hasta alen (F) {
      imprimir (“\nEl carácter “, ascii (pos_A + k – 1),
                “ aparece “, F [k], “ veces.”)
   }
fin
```

paramval()

Retorna un parámetro posicional que fue pasado al programa SL, ya sea desde la línea de comandos o desde el entorno de programación.

El valor retornado es siempre una cadena.

Vea también pcount().

Nota: Esta función es rara vez usada.

Sintaxis

```
sub paramval (p : numerico) retorna cadena
```

El valor de p debe estar entre 1 y pcount(); si no lo está, se retorna una cadena vacía.

Ejemplo

Imprime cada parámetro que se haya pasado al programa SL, uno por línea.

```
var
   k = 0
inicio
   imprimir ("\nLista de parámetros pasados al programa:")
   desde k=1 hasta pcount() {
      imprimir ("\n", k, " ", paramval(k))
   }
fin
```

pcount()

Retorna la cantidad de parámetros que fueron pasados al programa SL, ya sea desde la línea de comandos o desde el entorno de programación.

Vea también paramval().

Nota: Esta función es rara vez usada.

Sintaxis

```
sub pcount() retorna numerico
```

Ejemplo

Vea paramval().

pos()

Busca una cadena dentro de otra y retorna la posición en que aparece por primera vez una concordancia, si la misma se da.

El primer parámetro de la función es la cadena que posiblemente contiene la secuencia de caracteres que se desea buscar.

El segundo parámetro es lo que se desea buscar.

Opcionalmente puede especificarse una posición inicial para la búsqueda; si esto se omite, el compilador de SL genera código para buscar desde el primer carácter del primer parámetro.

Sintaxis

Alternativa 1: buscando desde el principio.

```
sub pos (donde_buscar  : cadena
         valor_buscado : cadena) retorna numerico
```

Alternativa 2: buscando desde cierta posición en adelante.

```
sub pos (donde_buscar  : cadena
         valor_buscado : cadena
         pos_inicial   : numerico) retorna numerico
```

Ejemplo

```
var
   prosa = "Cuando haya acabado esta guerra, y acabará pronto, ..."
inicio
   imprimir ("\n", pos (prosa, "acaba"))
   imprimir ("\n", pos (prosa, "paz"))
   /*
    * Debe encontrar la segunda aparición de la "acaba", pues la primera
    * inicia en la posición 13.
    */
   imprimir ("\n", pos (prosa, "acaba", 14))
fin
```

El programa anterior imprimirá:

```
13
0
36
```

random()

Genera un número pseudo-aleatorio cada vez que se la invoca.

El resultado es un entero mayor o igual a cero y menor a un valor tope que se pasa como parámetro.

Opcionalmente puede proveerse un segundo parámetro como “semilla” o valor inicial que se inyecta al generador de números pseudo-aleatorios. Si pasa este parámetro, asegúrese de hacerlo una sola vez en todo su programa. Nota: Se recomienda omitir esta “semilla”, ya que SL la inicializa automáticamente al iniciar la ejecución del programa.

Sintaxis

```
sub random (tope : numerico) retorna numerico
```

Ejemplo 1

El siguiente programa imprime 10 números elegidos “al azar”. Todos serán menores a 200. El cero podría aparecer.

Observe que cada vez que se lo ejecute, se obtendrán potencialmente diferentes valores.

```
var
   k = 0
inicio
   desde k=1 hasta 10 {
      imprimir (random (200), “ “)
   }
fin
```

Ejemplo 2

Suponga que desea simular el proceso de tirar 50 veces una moneda y obtener cuántas caras y cuántas cruces cayeron. El siguiente programa hace esta simulación usando `random()`.

Como `random()` utiliza una distribución uniforme, si se ejecutara este programa varias veces y se calculara un promedio de los resultados, se encontraría que las caras y cruces se distribuyen en una proporción igual o cercana a 50% cada una, como es de esperarse en un experimento de la vida real.

```
var
   cant_cara = 0
   cant_cruz = 0

   tirada = 0
   k = 0
inicio
   desde k=1 hasta 50 {
      /*
       * En este caso, random() producirá solo dos valores posibles: ceros y
       * unos. Asumamos que cero es cara y uno es cruz.
       */
      tirada = random (2)
      si ( tirada == 0 ) {
         inc (cant_cara)
      sino
```

```
            inc (cant_cruz)
        }
    }
    imprimir ("\nCantidad de caras: ", cant_cara,
              "\nCantidad de cruces:", cant_cruz)
fin
```

readkey()

Espera a que se pulse una tecla y retorna un código numérico que identifica lo pulsado por el usuario. El tiempo que se espera depende del parámetro, que está expresado en milisegundos.

Si se omite el parámetro, `readkey()` espera indefinidamente a que el usuario pulse una tecla. De contrario esperará el tiempo que se le indique (en milisegundos) , y si nada se pulsó en ese tiempo, retornará cero.

Por una decisión de diseño y portabilidad de SL, no se asigna ningún significado específico al valor retornado por esta función, aunque el usuario es libre de experimentar y observar qué valores se generan para cada tecla.

Si el dispositivo de entrada no es el teclado, porque por ejemplo es un archivo, retorna cero.

Sintaxis

```
sub readkey (milisegundos : numerico) retorna numerico
```

runcmd()

Ejecuta un programa externo cualquiera y retorna el estado de salida del proceso. El parámetro pasado a `runcmd()` incluye el nombre del comando y los argumentos que éste necesite. La ejecución del comando se realiza vía el procesador de comandos del usuario y las reglas de localización del comando externo son las de aquel.

El valor de retorno de `runcmd()` es el valor de salida, también llamado "estado de salida", del comando ejecutado.

Si la ejecución del comando no puede realizarse, `runcmd()` retorna 127. Nótese que este valor es arbitrario y bien podría ser que un comando se ejecute correctamente y retorne 127.

Sintaxis

```
sub rumcmd(cmd : cadena) retorna numerico
```

---

sec()

Retorna el tiempo transcurrido (en segundos) desde la medianoche del 1 de enero de 1970.

Esta función puede servir para medir el tiempo transcurrido de un momento a otro.

Nota: En versiones iniciales de SL esta función retornaba el tiempo transcurrido desde la medianoche.

Sintaxis

```
sub sec() retorna numerico
```

Ejemplo

El siguiente programa muestra cómo puede usarse `sec()` para observar cuánto tiempo llevó la ejecución de una parte del programa. En este caso, se mide cuánto tiempo llevó la ejecución repetida de una subrutina que invierte el contenido de un vector numérico.

```
var
   A : vector [50] numerico
   /*
    * t1 significa "tiempo 1" y t2 significa "tiempo 2", es decir,
    * contienen el valor de sec() en dos momentos diferentes.
    */
   t1, t2 : numerico
   k      : numerico
const
   CANT_REP = 251
inicio
   desde k=1 hasta alen (A) {
      A [k] = k*2
   }
   t1 = sec()
   /*
    * Repetiremos CANT_REP veces lo mismo solo para que tarde un poco y
    * podamos apreciar el tiempo transcurrido; de lo contrario es muy
    * rápido el proceso y el tiempo transcurrido es muy corto.
    */
   desde k=1 hasta CANT_REP {
      invertir_vec (A)
   }
   t2 = sec()
   imprimir ("\nEl vector A contiene:\n", A,
             "\n\nEl proceso de invertir el contenido de A llevó ", t2-t1,
             " segundos, repetido ", CANT_REP, " veces.")
fin


sub invertir_vec (ref A : vector [*] numerico)
```

```
var
   g = alen (A)
   k = 0
inicio
   desde k=1 hasta int(g)/2 {
      intercambiar (A [k], A [g-k+1])
   }
fin
```

set_color()

Permite cambiar los colores que se usarán al imprimir mensajes en la pantalla.

El segundo parámetro de `set_color()` indica el color de fondo que se desea. El primer parámetro (primer_plano) indica el color del texto mismo.

Si cualquiera de los parámetros es `0`, el color correspondiente no se ve afectado.

Por una decisión de diseño, los colores no tienen nombres o significados predefinidos, ya que al hacerlo se está limitando la posibilidad de portar SL a nuevas plataformas.

No tiene ningún efecto cuando el dispositivo de salida no es la pantalla.

Sintaxis

```
sub set_color (primer_plano, fondo : numerico)
```

set_curpos()

Posiciona el cursor en la línea y columna especificadas. La esquina superior izquierda de la pantalla corresponde a la posición (1, 1), es decir, línea 1 columna 1. Si solo se desea cambiar la columna conservando la fila actual, puede pasarse 0 como columna; similar efecto se logra pasando 0 como número de línea: solo se cambia la columna.

No tiene ningún efecto cuando el dispositivo de salida no es la pantalla.

Sintaxis

```
sub set_color (nueva_lin, nueva_col : numerico)
```

set_ifs()

Establece el separador de valores (o “campos”) a ser utilizado en subsiguientes llamadas a `leer()`.

Vea también `get_ifs()`.

`set_ifs()` debe ser llamado luego de cada `set_stdin()`, pues se restaura a su valor original (`“,”`) al cambiar el archivo de entrada.

Sintaxis

```
sub set_ifs (c : cadena)
```

Si c contiene más de un carácter, solo se usa el primero de ellos.

Ejemplo

Supóngase que desea leerse la cédula y el nombre y apellido de una persona. Entre el nombre y el apellido hay una coma.

Como la coma es por defecto para `leer()` el separador de campos, si no se cambia este separador, los valores no se leerán correctamente (solo se leerá hasta el nombre y el apellido quedará en el buffer de entrada). Usando `set_ifs()` puede superarse esta situación.

```
var
   cedula = “”
   nombre = “”
inicio
   set_ifs (“+”)
   imprimir (“\nIngrese Cédula, signo +, Nombre y apellido:”
   leer (cedula, nombre)
   imprimir (“\nCedula=”, cedula,
             “\nNombre y apellido=”, nombre)
fin
```

set_ofs()

Configura qué carácter utilizará `imprimir()` para separar los valores de los datos compuestos, como los registros y arreglos. Por defecto este carácter es una coma.

Además del carácter separador de valores, `set_ofs()` configura lo que se mostrará cuando `imprimir()` encuentre un arreglo que no está dimensionado. Por defecto se imprime “`<nodim>`”.

El primer carácter del parámetro de `set_ofs()` es el separador y lo que sigue es el indicador de “no dimensionado”, por lo que la configuración por defecto actúa como si se hubiese hecho `set_ofs(“,<nodim>”)` al iniciar el programa.

Se usa `get_ofs()` para obtener estos valores vigentes.

Sintaxis

```
sub set_ofs (c : cadena)
```

Ejemplo

El siguiente ejemplo imprime los datos de cada alumno en una línea de la pantalla. Si un alumno no tiene notas, se muestra “*Sin notas”. Los datos de un mismo alumno se separan unos de otros con un espacio.

```
tipos
   Alumno : registro {
               cedula : cadena
               notas : vector [*] numerico
            }
var
   lista : vector [5] Alumno
   k = 0
inicio
   lista = { {"1283912", {5, 4, 5, 5, 3}},
             {"3281242", {3, 2, 3, 1, 4}},
             {"1278217", {} },              // Este alumno no tiene notas
             {"2381923", {5, 5, 5, 5, 4}},
             {"1938281", {4, 3, 2, 2, 1}}
           }

   set_ofs (" *Sin notas")
   imprimir ("\nDatos de la clase:\n")
   desde k=1 hasta alen(lista) {
      imprimir (lista [k], "\n")
   }
fin
```

El resultado será:

```
Datos de la clase:
1283912 5 4 5 5 3
3281242 3 2 3 1 4
1278217 *Sin notas
2381923 5 5 5 5 4
1938281 4 3 2 2 1
```

---

set_stdin()

Establece desde donde se obtendrán los datos cada vez que se llame a `leer()`. Por defecto, se lee del teclado.

Si a `set_stdin()` se pasa un nombre de archivo que no existe, o no se tiene

permisos para leerlo, se devuelve el valor lógico FALSE.

Puede usarse "" (cadena vacía) como nombre de archivo para hacer que subsiguientes llamadas a `leer()` tomen los datos del teclado, cerrando así cualquier archivo que se haya abierto previamente.

`set_stdin()` restaura el separador de campos a "," (coma). Debe llamar a `set_ifs()` luego de `set_stdin()` si desea usar otro separador de campos.

Sintaxis

```
sub set_stdin (nombre_archivo : cadena) retorna logico
```

Ejemplo

El siguiente ejemplo lee el archivo "`resumen.txt`" y graba en el archivo "`letra_a.txt`" toda línea del primer archivo que contenga la letra "`a`", desplegando al final en la pantalla cuántas líneas fueron encontradas que cumplen el criterio señalado.

```
var
   linea       = ""
   cant_lineas = 0
const
   ARCH_ENTRADA = "resumen.txt"
   ARCH_SALIDA  = "letra_a.txt"
inicio
   imprimir ("\nEste programa muestra cómo usar set_stdin() para leer",
             " desde un archivo, y set_stdout() para enviar los",
             " resultados a otro archivo.\n")
   si ( not set_stdin (ARCH_ENTRADA) ) {
      imprimir ("\nNo se pudo abrir el archivo ", ARCH_ENTRADA,
                ".\nEl programa no puede continuar.")
      terminar("\nEjecución terminada con error.\n")
   }
   /*
    * Se desea leer una línea a la vez; por eso el separador es \n
    */
   set_ifs ("\n")
   si ( not set_stdout (ARCH_SALIDA) ) {
      terminar ("\nNo se pudo abrir el archivo " + ARCH_SALIDA)
   }
   /*
    * A partir de ahora, imprimir() grabará en el archivo abierto
    * más arriba y no mostrará nada en la pantalla.
    */
   leer (linea)
   mientras ( not eof() ) {
      si ( pos (linea, "a") > 0 ) {
         imprimir (linea)
         inc (cant_lineas)
      }
      leer (linea)
   }
   /*
    * Hacer que imprimir() muestre nuevamente los resultados en la pantalla.
    */
   set_stdout ("")
   imprimir ("\nFueron copiadas ", cant_lineas, " líneas\n")
fin
```

set_stdout()

Establece el destino de los datos que son generados a través de `imprimir()`. Puede ser un archivo o la pantalla, siendo este el destino por defecto.

Si a `set_stdin()` se pasa un nombre de archivo que no existe, éste es creado. Si el archivo ya existe, su contenido se pierde, a menos que se utilice la segunda sintaxis (ver más abajo) y se solicita que el contenido sea adicionado y no borrado.

Si el archivo no puede ser creado, porque no se tiene permisos por ejemplo, o porque ya está abierto, se retorna el valor lógico `FALSE`.

Puede usarse **“”** (cadena vacía) como nombre de archivo para hacer que subsiguientes llamadas a `imprimir()` envíen los datos a la pantalla, cerrando así cualquier archivo que se haya abierto previamente.

Sintaxis

Alternativa 1: El archivo se crea si no existe. Si ya existe, los datos previos se pierden.

```
sub set_stdout (nombre_archivo : cadena) retorna logico
```

Alternativa 2: Se especifica el “modo” en que se abrirá el archivo.

```
sub set_stdout (nombre_archivo : cadena
                modo_apertura  : cadena) retorna logico
```

Los valores posibles del parámetro “modo_apertura” son:

“wt” : El contenido previo del archivo se pierde. Si el archivo no existe, se crea. Equivale a no pasar este segundo parámetro.

“at” : Todo lo que se imprima se agregará al archivo, conservando su contenido previo. Si el archivo no existe, se crea.

Ejemplo

Vea el ejemplo de `set_stdin()`.

sin()

Calcula el seno de un ángulo expresado en radianes.

Sintaxis

```
sub sin (a : numerico) retorna numerico
```

sqrt()

Calcula la raíz cuadrada de un número.

El número cuya raíz cuadrada se desea calcular debe ser positivo. En caso contrario, se producirá un error de ejecución y el programa terminará.

Sintaxis

```
sub sqrt (n : numerico) retorna numerico
```

str()

Dado un valor numérico, produce la representación de ese mismo valor pero en formato de cadena. Varios aspectos de formateo pueden ser controlados por parámetros opcionales. Por ejemplo, la conversión puede tomar solo la parte entera, o incluir cierta cantidad de decimales, etc.

Se lo utiliza generalmente para mostrar un valor numérico con una cantidad precisa de dígitos decimales, más allá de lo que `imprimir()` muestra por defecto.

A menudo se escucha decir que `str()` "convierte" un valor numérico en cadena. Esto es técnicamente incorrecto, pero forma parte de la expresión cotidiana en programación. Es incorrecto porque el valor original permanece intacto. Lo que en realidad hace `str()` es producir una representación textual formateada del número.

La función `str()` siempre redondea (por exceso) el valor numérico. Esto quiere decir que si se tiene una variable numérica con valor `78.135`, al formatearlo con `str()` a dos decimales, se tendrá una cadena donde los últimos dos dígitos decimales serán `"14".`

Vea también `val().`

Sintaxis

Alternativa 1: Formateo por defecto: 2 decimales, ancho total no especificado y alineación a la izquierda.

```
sub str (n : numerico) retorna cadena
```

Alternativa 2: Formateo con 2 decimales pero con un ancho total determinado. En el ancho se debe contar el punto decimal, es decir, un ancho de 6 alanza para 3 dígitos enteros, un punto decimal y dos dígitos decimales. La alineación es a la derecha.

```
sub str (n     : numerico
         ancho : numerico) retorna cadena
```

Si la cantidad de dígitos de n es superior a ancho, se ignora el ancho y se retorna el resultado como si ancho haya sido 0. Este comportamiento se mantiene para todas las alternativas que acepten el parámetro ancho.

Alternativa 3: Formateo con un ancho determinado, pero indicando una cantidad específica de dígitos decimales. La alineación es a la derecha.

```
sub str (n        : numerico
         ancho    : numerico
         cant_dec : numerico) retorna cadena
```

Alternativa 4: Además de formatear con un ancho total definido y cantidad de decimales, se rellenará a la izquierda el resultado con cierto carácter, si el número formateado tiene menos dígitos que el ancho especificado.

```
sub str (n        : numerico
         ancho    : numerico
         cant_dec : numerico
         relleno  : cadena) retorna cadena
```

Ejemplo

El siguiente programa muestra algunos usos de str().

```
var
   n = 123.40451
   s = ""
inicio
   s = str (n);            imprimir ("\nAlternativa 1:",  s)
   s = str (n, 10);        imprimir ("\nAlternativa 2:",  s)
   s = str (n, 10, 3);     imprimir ("\nAlternativa 3:",  s)
   s = str (n, 0, 0);      imprimir ("\nAlternativa 3b:", s)
   s = str (n, 10, 0, "*"); imprimir ("\nAlternativa 4:",  s)
   s = str (n, 1, 1);      imprimir ("\nAlternativa 3c:", s)
fin
```

El resultado será:

```
Alternativa 1:123.40
Alternativa 2:    123.40
Alternativa 3:   123.405
Alternativa 3b:123
Alternativa 4:*******123
Alternativa 3c:123.4
```

strdup()

Dada una cadena y un número n, produce una nueva cadena que consta de n copias de la cadena original.

Sintaxis

```
sub strdup (c : cadena; n : numerico) retorna cadena
```

Ejemplo

El siguiente programa imprimirá 30 veces "[]".

```
inicio
   imprimir ("\n", strdup ("[]", 30))
fin
```

strlen()

Retorna la cantidad de caracteres de que consta una cadena.

La cadena vacía ("") tiene longitud 0.

Sintaxis

```
sub strlen (c : cadena) retorna numerico
```

Ejemplo

```
var
   m = "casa"
inicio
   imprimir ("\nLa palabra ", m, " tiene ", strlen (m), " letras")
fin
```

substr()

Hace una copia de una porción de una cadena, iniciando en cierta posición de carácter.

Si se omite el tercer parámetro, que indica cantidad de caracteres a copiar, se tomarán todos los caracteres de la cadena partiendo de la posición indicada por el segundo parámetro.

Si la posición inicial de copia es mayor a la longitud de la cadena, se retorna una cadena vacía.

Sintaxis

```
sub substr (c : cadena
            pos_inicial : numerico
            cant : numerico) retorna cadena
```

Ejemplo

```
var
   z = "ABCD"
inicio
   imprimir ("\nEl segundo carácter es ", substr (z, 2, 1))
   imprimir ("\nLos últimos tres son ", substr (z, 2))
   imprimir ("\nDesde la posición 5 solo hay vacío:", substr (z, 5, 5))
fin
```

swap()

Vea intercambiar().

tan()

Calcula la tangente de un ángulo, expresado en radianes.

Sintaxis

```
sub tan (a : numerico) retorna numerico
```

terminar()

Hace que el programa termine anticipadamente su ejecución.Note que el control NUNCA retorna al programa.

Puede proveerse un mensaje que se imprimirá antes de que termine la ejecución. Este parámetro es opcional; si no se especifica nada, se asume “”.

Sintaxis

```
sub terminar (mensaje : cadena)
```

Ejemplo

Vea el ejemplo dado para set_stdin().

upper()

Dada una cadena como parámetro, retorna una nueva cadena que consta de los mismos caracteres del parámetro, pero con aquellos que sean alfabéticos convertidos a mayúsculas.

Vea también lower().

Sintaxis

```
sub upper (c : cadena) retorna cadena
```

val()

Dado un valor de cadena que contiene dígitos de un número entero o decimal, produce un valor numérico que es almacenable en una variable de tal tipo, o que puede combinarse en expresiones numéricas.

A menudo se escucha decir que `val()` “convierte” una cadena a numérico. Esto es técnicamente incorrecto, pero forma parte de la expresión cotidiana en programación. Es incorrecto porque el valor original permanece intacto.

Si la cadena que se pasa como parámetro no contiene una secuencia que expresa un número, se retorna 0.

Vea también str().

Sintaxis

```
sub val (c : cadena) retorna numerico
```

Ejemplo

```
var
   z = “123.4”
inicio
   imprimir (“\n”, z, “ + 1 es ”, val (z) + 1)
   imprimir (“\n-12 * 2 es ”, val (“-12”)*2)
   imprimir (“\n'abc'+ 1 es “, val (“abc”)+1)
fin
```