Capítulo 6

Programación Interactiva

6.1 Cadenas

Una cadena es una secuencia de caracteres encerrados en comillas dobles.
Scheme proporciona varios procedimientos para manipular cadenas:

Procedimiento	Significado
string?	Evalúa si un argumento es una cadena
string-length	Proporciona la longitud de la cadena incluyendo espacios en blanco
string-append	Concatena las cadenas proporcionadas
subtring	Retorna la porción indicada de la cadena
symbol->string	Convierte un símbolo a una cadena
number->string	Convierte un número a una cadena
string=?	Evalúa la igualdad de las cadenas(distingue entre mayúsculas y minúsculas)
string-ci=?	Evalúa la igualdad entre cadenas(no distingue entre mayúsculas y minúsculas)

A continuación se presentan algunos ejemplos:

> (**string?** "Carlos")
#t
> (**string?** 'Carlos)
#f
> (**string-length** "Carlos Coello"
13
> (**string-append** "Carlos" " Coello")

"Carlos Coello"
> (**string-append** "This" " is a " "longer" " concatenation."
"This is a longer concatenation."
> (**substring** "Carlos Coello" 0 6)
"Carlos"
> (**substring** "Carlos Coello" 7 8)
"C"
> (**symbol->string** 'John)
"john"
> (**number->string** 5)
"5"
> (**string=?** "Carlos" "Carlos")
#t
> (**string=?** "carlos" "Carlos")
#f
> (**string-ci=?** "carlos" "Carlos")
#t
> (**string-ci?** "CARLOS" "carlos")
#t
> (**equal?** "Carlos" "carlos")
#f
> (**equal?** "Carlos" "Carlos")
#t

Advierta que **substring** retorna la parte de la cadena original que empieza con el índice dado *inicio* (definido por el primer entero proporcionado como argumento) cuya base es cero. Este procedimiento retorna pues todos los caracteres a partir de la posición indicada, pero sin incluir al correspondiente a la posicón *final* (el segundo entero proporcionado como argumento).

6.1.1 Problemas Resueltos

1. Defina un predicado **substring?** con dos parámetros, **sstr** y **strng** que evalúe si la cadena **sstr** es una subcadena de **strng**. Pruebe su predicado con los siguientes ejemplos:

   (**substring?** "a es u" "Esta es una cadena.") $\Longrightarrow$ **#t**

(**substring?** “anillo” “Esta es una cadena.”) $\Longrightarrow$ **#f**
(**substring?** “” “Esta es una cadena”) $\Longrightarrow$ **#t**

El procedimiento es el siguiente:

```
(define substring?
  (lambda (sstr strng)
    (substring?-aux sstr strng
(- (string-length strng) (string-length sstr)))))
```

El procedimiento **substring?-aux** es el siguiente:

```
(define substring?-aux
  (lambda (sstr strng acc)
    (cond
      ((zero? acc) #f)
      (else (or (string=? sstr (substring strng acc
        (+ acc (string-length sstr))))
(substring?-aux sstr strng (sub1 acc)))))))
```

2. Defina un procedimiento **invierte-string** que tome como argumento a una cadena y retorne dicha cadena con sus caracteres en orden reverso. El siguiente procedimiento podría serle de utilidad:

```
(define substring-ref
  (lambda (strng n)
    (substring strng n (add1 n))))
```

Pruebe su procedimiento con:

(invierte-string “Jack y Jill”) $\Longrightarrow$ **“lliJ y kcaJ”**
(invierte-string “padre y madre”) $\Longrightarrow$ **“erdam y erdap”**
(invierte-string “”) $\Longrightarrow$ **“”**

Los procedimientos son los siguientes:

(**define** invierte-string
(**lambda** (strng)
(reverse-aux strng (**string-length** strng))))

(**define** reverse-aux
(**lambda** (strng n)
(**cond**
((**zero?** n) “”)
(**else** (**string-append** (substring-ref strng (sub1 n))
(reverse-aux strng (sub1 n)))))))

3. Una cadena es un palíndroma si su reverso es igual que la cadena original. Por ejemplo ”radar” y ”oso” son palíndromas. Defina un predicado **palindroma?** que evalúe si una cierta cadena dada es o no un palíndroma. Pruebe su predicado con:

   (**palindroma?** “arroz daba abad zorra”) $\Longrightarrow$ **#t**
   (**palindroma?** “radar y maiz”) $\Longrightarrow$ **#f**

   El procedimiento es el siguiente:

   (**define** palindrome?
   (**lambda** (strng)
   (**string=?** strng (string-reverse strng))))

6.2 Begin Implícito

El uso de **begin** nos permite evaluar varias expresiones de manera secuencial y retornar el valor de la última de ellas. Esto es útil si tenemos una expresión **if** que espera sólo una expresión en cada una de sus cláusulas. Hay ciertas formas especiales en Scheme que tienen un **begin** implícito construido dentro de su definición (p.ej., **lambda**, **let**, **letrec** y **cond**). Por ejemplo:

(**lambda** (a b)
(**writeln** a)
(**writeln** b)
(* a b))

es lo mismo que

```
(lambda (a b)
  (begin
    (writeln a)
    (writeln b)
    (* a b))
```

También,

```
(let ((a 1) (b 4))
  (writeln a)
  (writeln b)
  (* a b))
```

es lo mismo que

```
(let ((a 1) (b 4))
  (begin
    (writeln a)
    (writeln b)
    (* a b))
```

6.3 Entrada y Salida

Scheme tiene los siguientes procedimientos para entrada y salida:

Procedimiento	Significado
display	Imprime sin usar comillas dobles
newline	Mueve el cursor al inicio de la línea siguiente
write	Imprime con comillas dobles
read	Se usa para introducir información a un programa

El procedimiento **read** normalmente se usa como un procedimiento sin argumentos (llamado *thunk* en inglés) que se invoca encerrando entre paréntesis el nombre de la variable.

```
(define read-demo
(lambda ()
(display "Introduzca datos (escriba ACABE para terminar): ")
(let ((response (read)))
(cond
((eq? response 'ACABE) (display "Gracias. Adios."))
(else (display "Introdujo: ")
(write response)
(newline)
(read-demo))))))
```

Ahora, veremos un procedimiento que determina si un cierto número se imprime o no.

```
(define numeros-primos
    (lambda ()
(display "Deme un numero: ")
(let ((n (read)))
 (display (string-append "El numero " (number->string n)))
    (if (< n 2) (display " no es valido.")
    (if (check-primo n 2)
(display " es un primo.") (display " no es un primo.")))
    (newline)
    (display "Deseas dar otro numero? ")
    (let ((response (read)))
(if (eq? response 'si)
    (numeros-primos)
    (begin
    (display "Eso es todo.")
    (newline)))))))
(define check-primo
    (lambda (n k)
    (cond
((= k n) #t)
(else (and (not (zero? (remainder n k)))
 (check-primo n (add1 k)))))))
```

6.4 Problemas Propuestos

1. Defina un procedimiento **acronimo** que tome una cadena **strng** y regrese la siglas que corresponden al título dado. Para hacer el probema más interesante deberá ignorar las preposiciones **at**, **the**, **in**, **of**, **and**, **for** & **with** en la generación del acronimo. Evalúe su procedimiento con:

   (**acronym** "The United States of America") $\Longrightarrow$ "USA"
   (**acronimo** "Association for Computing Machinery") $\Longrightarrow$ "ACM"
   (**acronimo** "Structure and Interpretation of Computer Programs") $\Longrightarrow$ "SICP"
   (**acronimo** "Tools with Artificial Intelligence") $\Longrightarrow$ "TAI"
   (**acronimo** "University of Alabama in Huntsville") $\Longrightarrow$ "UAH"
   (**acronimo** "University of California at Los Angeles") $\Longrightarrow$ "UCLA"
   (**acronimo** "Quod Erat Demonstrandum") $\Longrightarrow$ "QED"

2. Escriba un programa que pida al usuario una cantidad de dinero. Por ejemplo:

   ```
   ‘Para qué cantidad desea obtener cambio?  $
   ```

   El usuario deberá escribir una cantidad tal como 23.45. El programa dirá entonces cómo puede conformarse este monto en billetes de 100 pesos, de 20 pesos, de 10 pesos, de 5 pesos y de 1 pesos. Así mismo, considerará el uso de monedas de 25 centavos, de 10 centavos, de 5 centavos y de 1 centavo. La salida deberá decir algo como lo siguiente:

   ```
   Tu cambio es:
   1 billete de 20 pesos
   3 billetes de 1 peso
   1 moneda de 25 centavos
   2 monedas de 10 centavos
   ```

   El programa deberá preguntar al usuario si desea obtener cambio para otra cantidad. Si la respuesta es "no", entonces deberá terminar la ejecución del programa.

3. Un tal reverendo Zeller desarrolló una fórmula para calcular el día de la semana para cualquier día dado del calendario Gregoriano. Las entradas del algoritmo se especifican de la siguiente manera:

   - m es el mes del año, considerando a Marzo con $m = 1$. Enero y Febrero son los meses 11 y 12 del año anterior.
   - d es el día del mes.
   - y es el año del siglo.
   - c es el siglo *anterior*.

   Por ejemplo, para el 4 de Julio de 1989, $m = 5$, $d = 4$, $y = 89$, y $c = 19$, mientras que el 25 de enero de 1989 se codifica como $m = 11$, $d = 25$, $y = 88$, y $c = 19$. El algoritmo para calcular el día de la semana es el siguiente:

   1. Tomar la parte entera de $(13m - 1)/5$. Llamar a esto a.
   2. Tomar la parte entera de $y/4$. Llamar a esto b.
   3. Tomar la parte entera de $c/4$. Llamar a esto e.
   4. Calcular $f = a + b + e + d + y - 2c$.
   5. Hacer que r sea igual a f modulo 7.
   6. r nos indica el día de la semana. Si $r = 0$, se trata de Domingo, si $r = 1$ se trata de Lunes y así sucesivamente.

   Escriba un programa llamado **dia-semana** que pida el mes, día y año. El mes debe introducirse de manera habitual: Enero es el mes 1 y Diciembre es el mes 12. El ao también debe ingresarse de manera normal (p.ej., 2001). El programa debe convertir estos datos en lo que necesite el algoritmo y calcular el día de la semana. La salida debe ser en el formato "14/12/2001 es un Viernes". El programa debe preguntar al usuario si desea ingresar otra fecha. Si el usuario responde que "no", entonces el programa deberá terminar.

   (**dia-semana**)
   Introduzca el mes (1..12): 2

Introduzca el dia (1..31): 24
Introduzca el año: 2005
2/24/2005 es un Jueves
Quieres dar otra fecha? si

Introduzca el mes (1..12): 5
Introduzca el dia (1..31): 10
Introduzca el año: 1980
5/10/1980 es un Sabado
Quieres dar otra fecha? si

Introduzca el mes (1..12): 1
Introduzca el dia (1..31): 13
Introduzca el año: 1989
1/13/1989 es un Viernes
Quieres dar otra fecha? si

Introduzca el mes (1..12): 10
Introduzca el dia (1..31): 9
Introduzca el año: 1995
10/9/1995 es un Lunes
Quieres dar otra fecha? no

Eso es todo

4. Escriba un procedimiento interactivo llamado **numero-perfecto** que lea un número y determine si es perfecto o no. Un número es **perfecto** si la suma de todos sus factores, incluyéndose él mismo, genera el número mismo. La salida debe ser exactamente comose muestra a contiuación.

(**numero-perfecto**)
Dame un numero: 10
Los factores de 10 son: 1 2 5
El numero 10 no es perfecto.
Quieres dar otro numero? si

Dame un numero: 6
Los factores de 6 son: 1 2 3
El numero 6 es perfecto.
Quieres dar otro numero? si

Dame un numero: 28
Los factores de 28 son: 1 2 4 7 14
El numero 28 es perfecto.
Quieres dar otro numero? si

Dame un numero: 496
Los factores de 496 son: 1 2 4 8 16 31 62 124 248
El numero 496 es perfecto.
Quieres dar otro numero? no

Bye, bye.