Paradigmas de Programación
Paradigma Funcional

Programación en Scheme

Resolver los siguientes problemas.

1. Hallar el factorial de un número entero.

2. Encontrar el máximo elemento de una lista.

3. Hallar la longitud de una lista.

4. Sumar los elementos de una lista.

5. Determinar si un elemento es miembro de una lista.

6. Concatenar dos listas.

7. Hallar el n-ésimo elemento de una lista.

8. Determinar si una lista es sublista de otra. (tener en cuenta que los elementos de una lista pueden ser a su vez listas).

9. La lista ´(raiz, izq, der) representa un árbol binario donde la rama izquierda y la rama derecha representan a su vez árboles binarios y los nodos hojas se representan con la lista vacía.

10. Un árbol binario de nivel *n* es *completo, si "cada nodo de nivel n es una hoja y cada nodo de nivel menor que n no tiene subárboles izquierdo y derecho vacíos"*
Definir una función de un argumento para determinar si el árbol dado como argumento es completo.

11. Recorrer un árbol en preorden, formando una lista con los nodos que se visitan.

12. Idem 10) en inorden.

13. Idem 10) en posorden.

14. Dadas dos listas, determinar si una es prefijo de la otra.

15. Dadas dos listas, determinar si una es sufijo de la otra.

16. Dados un árbol y un elemento cualquiera, determinar si el elemento pertenece al árbol.

17. Dada la serie de Fibonacci 1, 1, 2, 3, 5, 8, 13, 21, ...
Realizar una función para calcular el n-ésimo elemento de la serie.

18. Dada una lista que con elementos duplicados, obtener otra sin elementos duplicados.

19. Dados dos elementos X e Y y una lista, obtener otra lista en la cual todas las ocurrencias de X hayan sido reemplazadas por Y.

20. Dados un elemento y una lista, obtener dos sublistas. Una de las sublistas debe contener todos los elementos menores al elemento dado y la otra, los mayores.

21. Dados una lista y dos elementos, determinar si estos son consecutivos.

22. Dada una lista cuyos elementos pueden ser a su vez sublistas, obtener otra lista que sólo tenga elementos atómicos. Por ejemplo dada la lista ‘(a b (c d) e), al aplicar la función obtener como resultado la lista ´(a b c d e).

23. Dada la siguiente declaración de hechos:

version(miprograma, 0)
version(miprograma, 1)
version(miprograma, 2)
version(miprograma, 3)
version(miprograma, 4)

Definir una función mayor de un argumento para determinar la mayor versión de miprograma. El argumento de la función es una lista que representa la información declarada en los hechos.

24. Desarrollar en Scheme, la función reversa de un argumento que, dada una lista de entrada, genere la lista reversa correspondiente. La lista de entrada es una lista binaria donde el primer elemento es un átomo y el segundo es una lista de elementos anidados, salvo en el último nivel, que consiste de una lista de un solo elemento. La lista revertida tiene la misma estructura que la lista de entrada en cuanto al número de anidamientos, pero el orden de los elementos de la lista de entrada ha sido invertido en cuanto al nivel de profundidad.
Por ejemplo,
(reversa ‘(a (b (c)))) => (c (b (a)))

25. Definir en Scheme una función que realice la asignación de las materias a dictar con las aulas disponibles para un día dado. La unidad académica tiene programadas sus actividades en módulos horarios fijos: *mod1*, *mod2*, *mod3*, *mod4*, *mod5*.
Cada *aula* posee una capacidad de alumnos que puede alojar.
Cada m*ateria* posee el número de alumnos que la cursan (se le deberá asignar un *aula* de capacidad igual o mayor) y en qué *módulo horario* de dicta (uno de los módulos indicados mod_i).
Se le pide que defina la función *asignar* de dos argumentos materias y aulas.

Al ser evaluada *asignar* deberá retornar como resultado una lista con pares materia-aula. Se supone que siempre se evalúa con un par de argumentos *materias, aulas* para el cual hay solución.
materias: es la lista de materias. Ésta podría ser una lista de listas, en la cual cada sublista contenga el nombre de la materia, número de alumnos que cursan, *modulo horario* en que

se dicta, y aula asignada (si no se ha asignado aún aula se podría colocar un cero o algún símbolo que identifique esta situación).

aulas: es la lista de aulas. Ésta podría ser una lista de listas, en donde las sublistas contengan nombre de aula, capacidad y los módulos horarios: *mod1 mod2 .... mod5*. Si ya se ha asignado *materia* a un dado módulo horario, en lugar del nombre del módulo horario, podría aparecer el nombre de la materia asignada.

26. Definir en Scheme una función productoPolinomio con una lista de listas como argumento, y que su evaluación de cómo resultado una lista de listas, en donde cada una de las listas está formada por un elemento de las listas componentes del argumento. Por ejemplo,

(productoPolinomio '((a s d) (1 2) (# $ &))) =>
'((a 1 #) (a 1 $) (a 1 &) (a 2 #) (a 2 $) (a 2 &) (s 1 #) (s 1 $) (s 1 &) (s 2 #) (s 2 $) (s 2 &)
(d 1 #) (d 1 $) (d 1 &) (d 2 #) (d 2 $) (d 2 &))

En el ejemplo se observa que cada una de las sublistas del resultado tiene un elemento de cada una de las tres sublistas del argumento original. A continuación se presenta otro ejemplo:

(productoPolinomio '((1 2) (a t c))) => '((1 a) (1 t) (1 c) (2 a) (2 t) (2 c))

27. Si *f* es una función numérica y *n* un número entero positivo, luego podemos construir una aplicación de *f* repetida *n* veces, la cual se define como la función que al ser evaluada en *x* es igual a *f(f(.....(f(x))...)*. Si *f* es la operación de hallar el cuadrado de un número, luego la aplicación repetida *n* veces de *f* es la función que eleva su argumento a la potencia 2^n. Definir en Scheme, una función que tenga dos argumentos: una función *f* y un entero positivo *n,* y retorne como resultado una función que aplica *n* veces la función *f*.
Por ejemplo,
(define cuadrado (lambda (y) (* y y))) => cuadrado
((repetir cuadrado 2) 5) => 625

28. Definir en Scheme, una función de dos argumentos, una *lista* y un número entero positivo *n*, de tal forma que al ser evaluada retorne como resultado la lista original, en la cual los primeros *n* elementos se han colocado al final de *lista*. Por ejemplo,
(moverAtras '(a s d 3 4 5) 2) => (d 3 4 5 a s)
(moverAtras '(d d 2 3 7 a a) 3) => (3 7 a a d d 2)

29. Definir en Scheme, una función sust de tres argumentos, el primero es *Lista1* (una lista), el segundo es *Patron1* (lista), el tercero es un *Patron2* (lista). La función sust al ser evaluada devolverá como resultado la lista resultante de sustituir todas las ocurrencias de *Patron1* en *Lista1* por *Patron2*.
(sust '(a b c d 1 a b c 1 a) '(1 a) '(c)) => (a b c d c b c c)
(sust '(1 2 s d a a 1 d f a a 1 d f) '(a a 1) '(c c c c)) => (1 2 s d c c c c d f c c c c d f)

30. Se esta construyendo un editor de texto en Scheme. En este programa un fichero es una lista compuesta por sublistas en la cual cada línea es una sublista, cuyo primer elemento es su posición en el fichero y el segundo otra lista con la cadena de caracteres que la compone.
((1 (Esta es la primer linea))
(2 (del fichero que se esta desarrollando.))
(3 (+ 1 2 .....))
(... ....)
)

Se le solicita que defina las funciones Scheme:
(Adicionar Fichero Linea Posición)
(Borrar Fichero Posición)
En estas funciones *Fichero* es una lista como la indicada anteriormente, *Posición* un entero que indica la ubicación de la *Línea* a eliminar o adicionar. La evaluación de estas funciones deberá dar como resultado el *Fichero* modificado. Tenga en cuenta que en ambos casos el *Fichero* deberá tener su líneas con el número que indica su posición actualizado con la nueva estructura.

31. Se le solicita que proponga las funciones Scheme PUTPROP y GETPROP.

PUTPROP incorpora una terna (objeto atributo valor) en una lista BasedeDatos, en este caso considerar que si existía previamente alguna terna con los valores de *objeto* y *atributo* que se quieren incorporar, se debe eliminar la misma y luego incorporar la nueva terna;

```
(define PUTPROP
        (lambda (BasedeDatos Objeto Atributo Valor)
        .....)
```

GETPROP obtiene el valor para un conjunto (objeto atributo)

```
(define GETPROP
         (lambda (BasedeDatos Objeto Atributo)
        .....)
```

Por ejemplo, considere el siguiente escenario:

- (define base '((juan edad 30) (juan direccion (lavaise 610)) (pedro edad 34) (pedro hijos 2) (pedro sueldo 1400)))

base

- (GETPROP base 'juan 'edad)

30

- (GETPROP base 'pedro 'hijos)

2

- (define base (PUTPROP base 'luis 'direccion '(sastre 701)))

((luis direccion (sastre 701)) (juan edad 30) (juan direccion (lavaise 610)) (pedro edad 34) (pedro hijos 2) (pedro sueldo 1400))

- (define base (PUTPROP base 'pedro 'edad 36))

((pedro edad 36) (luis direccion (sastre 701)) (juan edad 30) (juan direccion (lavaise 610)) (pedro hijos 2) (pedro sueldo 1400))

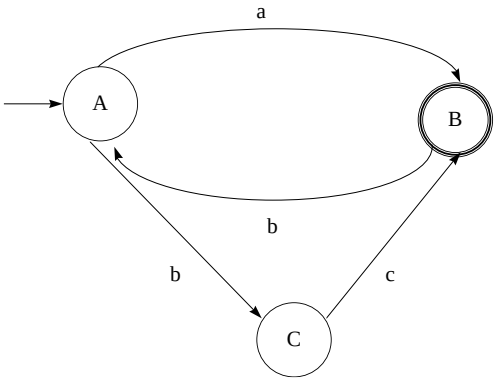
32. Un Aceptador de Estados Finitos (AEF) es un grafo dirigido que representa un LENGUAJE (conjunto de strings). Los nodos de este grafo representan ESTADOS y los arcos TRANSICIONES entre estados. Una string se dice que es ACEPTADA por el AEF, si mediante la misma es posible evolucionar desde el ESTADO INICIAL del aceptador hasta el ESTADO FINAL del mismo. A continuación se representa como ejemplo el AEF denominado M1, el cual puede aceptar las strings:

aba ; bcbaba ; …

con las siguientes secuencias de estados en el ACEPTADOR

$A \rightarrow (a) \rightarrow B \rightarrow (b) \rightarrow A \rightarrow (a) \rightarrow B$

$A \rightarrow (b) \rightarrow C \rightarrow (c) \rightarrow B \rightarrow (b) \rightarrow A \rightarrow (a) \rightarrow B \rightarrow (b) \rightarrow A \rightarrow (a) \rightarrow B$

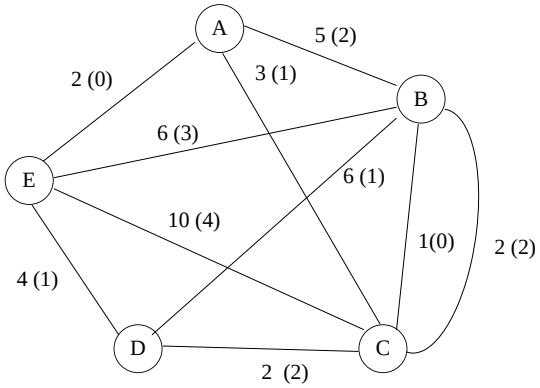


M1 : Aceptador - A: estado inicial - B: estado final

Se le solicita que defina la función en scheme ACEPTADOR, la cual debe poseer dos argumentos GRAFO y STRING. GRAFO representa el aceptador de estados finitos y STRING es la lista con la cual se representa la string que se desea analizar. Si STRING permite llegar desde el estado inicial al final, la función ACEPTADOR devolverá como resultado de su evaluación TRUE. Para simplificar el problema se supondrá que en el ACEPTADOR no es posible que exista más de un arco que salga de un nodo etiquetado con el mismo símbolo.

Nota: Describa claramente la representación que utilizará para el grafo. Puede definir todas las funciones auxiliares que requiera.

33. Dado un mapa como el siguiente, en el cual se indican las ciudades, las distancias entre ellas y cuantos kilómetros son de camino de tierra (la cantidad entre paréntesis). Construir un programa en Scheme, cuya función principal sea caminoElegido



(caminoElegido ***CiudadI CiudadF MáximoCaminoTierra*****)**

MáximoCaminoTierra**:** máximo camino de tierra tolerado en la ruta de mínima longitud entre CiudadI y CiudadF.

La evaluación de la función caminoElegido dará como resultado la lista Ruta que contiene el listado de ciudades que hay que recorrer para ir de CiudadI a CiudadF por el camino de mínima longitud, y que por el mismo no sea necesario transitar una longitud total de camino de tierra que supere MáximoCaminoTierra.

34. Existen muchas aplicaciones en las cuales se utilizan matrices de dos dimensiones de gran tamaño. Si la mayoría de los elementos de la matriz son ceros, la matriz se denomina Matriz Rala. Obviamente una Matriz Rala puede ser más eficientemente representada si se alamcenan en memoria sólo las celdas con valores distintos de cero. Con tal motivo se le solicita que defina una función Scheme *Convierte,* cuyo único argumento es la matriz completa, y que como resultado de su evaluación retorna la matriz ingresada, en una representación compacta. El argumento es una matriz representada por una lista de listas, en donde cada una de las sublistas representa las respectivas filas de la matriz.

Por ejemplo:

A :=: ((10 0 0 10 0) (0 20 0 0 0) (30 0 0 0 30) (40 0 0 0 0) (50 0 0 0 50))

es una matriz de 5x5, en donde A[1,1] = 10 ; A[1,2] = 0 ; A[2,3] = 0 ; A[5,5] = 50

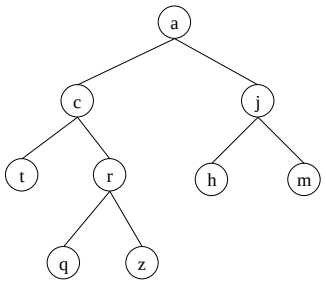
Convierte deberá retornar la matriz representada por una lista de listas, en donde las sublistas representarán cada uno de los elementos distintos de cero.

Las sublistas tendrán tres elementos: (*fila columna valor*).

- (*Convierte* '((10 0 0 10 0) (0 20 0 0 0) (30 0 0 0 30) (40 0 0 0 0) (50 0 0 0 50)))

((1 1 10) (1 4 10) (2 2 20) (3 1 30) (3 5 30) (4 1 40) (5 1 50) (5 5 50))

35. Uno de los recorridos de árboles es en forma horizontal, el cual consiste en visitar los nodos que están en un mismo nivel (de izquierda a derecha), luego proceder con los nodos del próximo nivel y así sucesivamente hasta visitar todos los nodos del árbol. A continuación se presenta un ejemplo de recorrido horizontal.



(a c j t r h m q z)

El diseño de una función recursiva para este recorrido no es inmediata, ya que el mismo no sigue la estructura recursiva del árbol. Sin embargo no es imposible encontrar un esquema que lo haga posible.

El problema puede pensarse de la siguiente forma (consideremos el ejemplo). Tras tomar la raíz, nos quedan dos subarboles por recorrer. Si a continuación tomamos el nodo **c**, nos

quedarán por recorrer sus dos subárboles a continuación del subárbol con **j** como raíz. Si a continuación tomamos la raíz **j**, sus dos subárboles quedarán pendientes a continuación de los dos subarboles de **c**. El proceso continua hasta haber recorrido todos los nodos. Por lo tanto el proceso podría pensarse a partir de una lista de árboles, y a medida que se extrae una raíz, los árboles generados se incorporan al final de la lista. Faltaría analizar el caso de los nodos hojas, aunque los mismos no incorporan ninguna complejidad.
Se le pide que defina en Scheme la función

(recorreHorizontal *árbol*)

que retorna como resultado una lista con los nodos del árbol ordenados de acuerdo a un recorrido horizontal.
Deberá indicar claramente para cada función que defina: objetivo, significado de los argumentos, y resultado que retorna como resultado de su evaluación.
Describa además que estructura utilizará para representar los árboles.

36. Una función y = f(x), la cual representa un polinomio de grado *n*, puede ser expresada en Scheme por medio de una lista ternaria donde el primer elemento es el operador (+, -, *, ^ (potencia), /), y el segundo y tercer elemento cada operando de dicha operación (pudiendo ser estos operaciones por evaluar).

Ejemplos de polinomios serían los siguientes:

Notación Formal	**Representación en Scheme**
y = f(x) = x + 4	(define y (list + ‘x 4))
y = f(x) = x^2 + x - 3	(define y (list - (list + (list expt 'x 2) 'x) 3))
y = f(x) = x^4 + x^3 - x	(define y (list - (list + (list expt 'x 4) (list expt 'x 3)) 'x))

Se le solicita definir una función de dos argumentos en Scheme denominada

(FuncionImpar función lista)

Donde :

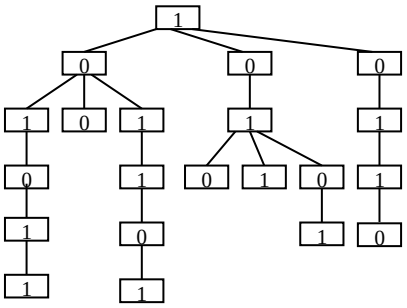
función: es una lista que representa un polinomio con las características antes mencionadas.
lista: es una lista de números (x1 x2 x3 x4 … xn) donde cada x_i es un número perteneciente al dominio de la función *función*

La evaluación de ***FuncionImpar*** retornara #***true*** en caso de para todos los x_i de la lista de números se cumpla que ***función***(x_i) = - ***función***($-x_i$), caso contrario se evaluación será #**false.**

Nota:

Obviamente que deberá definir una *función auxiliar* que permita la evaluación de una función con las características antes mencionadas para un número dado.

37. Un árbol como el que se presenta en la figura representa cadenas de un lenguaje binario donde cada palabra (secuencia de bits) esta representada por una rama del mismo.



En Scheme este tipo de árbol se puede representar por medio de una lista, donde el primer elemento es el valor del nodo y los restantes sus hijos correspondientes. (Que a su vez pueden ser padres de cero, uno o más árboles).

Se le solicita que defina la función (**check** árbol palabra), donde:

árbol : es un árbol con las características antes mencionadas.

palabra : es una lista (no vacía) donde cada elemento es un 0 o 1.

Esta función al ser evaluada retornará #True en caso que la palabra sea una cadena del lenguaje que representa el árbol argumento, caso contrario retornara #False.

Para el árbol de la figura:

(**check** arbol '(1 0 1 1))

#True

(**check** arbol '(1 0 1 0 1))

#True

Indique claramente cual es la representación utilizada para el árbol.

38. Se le solicita que defina la función Scheme

(mapeoFunciones *ListaFunciones Lista*)

en donde *ListaFunciones* es una lista de funciones de un argumento. La aplicación de **mapeoFunciones** aplica todas las funciones de *ListaFunciones* a los elementos de Lista, retornando una lista con los resultados de estas evaluaciones.

El resultado final es una lista de listas, en donde cada sublista contiene los resultados de aplicar las funciones de *ListaFunciones* a cada elemento de *Lista*.

Por ejemplo

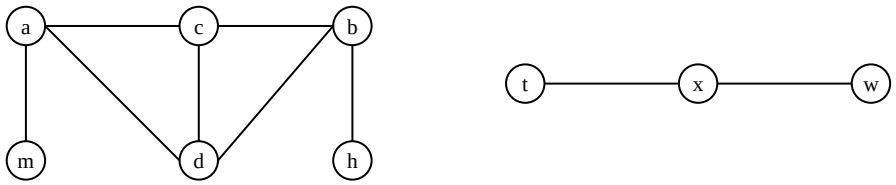
- (define suma1 (lambda (x) (+ 1 x)))
  → suma1
- (define resta1 (lambda (x) (- x 1)))
  → resta1
- (define por2 (lambda (x) (* x 2)))
  → por2
- (define a (cons suma1 (cons resta1 (cons por2 ( )))))
  → (<procedure suma1> <procedure resta1> <procedure por2>)
- (mapeoFunciones a '(2 2 3))
  → ((3 1 4) (3 1 4) (4 2 6))

Nota: Tener en cuenta que cada sublista consiste en los resultados de aplicar todas las funciones a un único elemento de *Lista*.

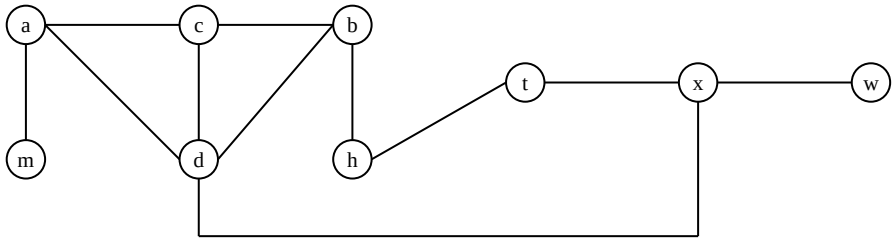
39. Un grafo G = {V, A}es conexo si para todo par vértices $v_i, v_j \in V / v_i \neq v_j$ existe un camino (secuencia de aristas $\in$ A) que los une.

Ejemplos

Grafo no conexo



Grafo conexo



Definir una función en Scheme

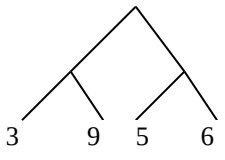
(**EsConexo** Grafo)

la cual al ser evaluada dará como resultado #True si el argumento Grafo corresponde a un grafo conexo, caso contrario retornara #False.

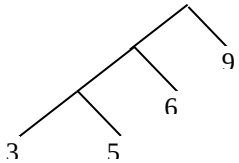
40. Sea $p_1,p_2,...p_n$ un conjunto de números positivos los cuales reciben el nombre de pesos y A un árbol binario con n hojas (n > 1).

Si asignamos a cada hoja del árbol un peso, obtenemos lo que se denomina un *árbol binario para los pesos $p_1,p_2,...p_n$.*

La siguiente sumatoria $\sum_{i=1}^{n} p_i * h(p_i)$ da el peso total del árbol, donde $h(p_i)$ es él número de nivel (cantidad de aristas para llegar a una hoja) asignado al peso p_i. A continuación se dan dos ejemplos de peso de árbol:



Árbol_1
P(Arbol_1) = 46



Árbol_2
P(Arbol_2) = 45

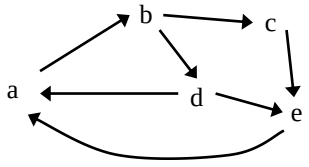
Definir una función en Scheme denominada ***mapeoarboles*** la cual recibe como argumento una lista, donde cada elemento es una árbol binario con las características antes mencionadas, y retorna luego de su evaluación el árbol de menor peso.

Nota

Obviamente que deberá definir una función para evaluar el peso de cada árbol.

Indique claramente la estructura de datos que emplea para representar un árbol binario.

41. Dado el grafo dirigido representado en la siguiente figura:



Una representación para el mismo en Scheme podría consistir en una lista que contenga dos sublistas: una representando el conjunto de nodos y otra representando el conjunto de arcos, donde cada arco es a su vez una lista de dos elementos, el nodo inicial y el final. Por ejemplo:

(A) ((a b c d e) ((a b) (b c) (b d) (c e) (d a) (d e) (e a)))

Otra representación válida consiste en una lista de listas, donde cada sublista contiene como elementos un nodo y una lista de los nodos hacia los cuales éste está conectado, por ejemplo:

(B) ((a (b)) (b (c d)) (c (e)) (d (a e)) (e (a)))

Note que el elemento (d (a e)) representa la existencia de los arcos (d a) y (d e) en la representación anterior.

Se le pide que desarrolle una función que permita obtener la representación **(B)** de un grafo dado en la representación **(A)**.

42. Se quiere calcular las comisiones de viajantes de una empresa. Las mismas depende de los montos vendidos por cada viajante y del producto que vendió. Para ello se cuenta con la información de las ventas por producto que realizó cada viajante y con los porcentajes de comisiones de cada producto. Se le pide que defina las funciones en scheme necesarias para calcularlas sabiendo que la información sobre viajantes está organizada de la siguiente manera:

((viajante1(venta,productoA)(vta,productoB)....) (viajante2(venta, productoC)(venta,ProductoA)..)...)

y la información sobre los productos está organizado en otra lista de la siguiente manera:

((productoA, porcentajeA) (produtoB, porcentajeB).....)

El resultado de la evaluación de la función que debe crear, debe ser la lista de los viajantes a la que se le agrega un dato más al final que es su comisión. De esta manera la lista resultante será:

((viajante1(vta prodA)(vta prodB)..... comisión)(....))

43. Se tiene la siguiente gramática que permite representar un subconjunto de funciones escalares:

Dígito =:= 0|1|2|3|4|5|6|7|8|9

Número =:= <Dígito> | <Número> <Dígito>

Función =:= (^ x <Número>) | (+ <Función> <Función>) | (- <Función> <Función>) |
(* <Número> <Función>) | (Seno(x)) | (Coseno (x))

Como se puede observar esta gramática genera funciones de la forma *(Operador Operando1 Operando2)* (excepto para Seno(x) y Coseno(x)).

En Scheme las siguientes funciones tendrían la siguiente representación:

f(x) = $x^3 + 3x^5 - Seno(x)$	(define f '(+ (^ x 3) (- (* 3 (^ x 5) (Seno(x))))))
F(x) = 3 Seno(x)	(define f '(* 3 (Seno(x)))
F(x) = Coseno(x) + x - 1	(define f ' (+ (Coseno(x)) (- x 1)))

Se le solicita que defina la función ***IntegralDefinida*** (función x1 x2), la que al ser evaluada retorna el valor de $\int_{x1}^{x2} funcion\ dx$

Reglas de Integración a tener en cuenta

$\int x^n dx = \frac{1}{n+1} x^{n+1}$	$\int (f(x)+g(x))dx = \int f(x)dx + \int g(x)dx$	$\int (f(x)-g(x))dx = \int f(x)dx - \int g(x)dx$
$\int Kf(x)dx = K\int f(x)dx$	$\int Seno(x)dx = Co\,\text{sen}\,o(x)$	$\int Co\,\text{sen}\,o(x)dx = -Seno(x)$

Nota: a cada resultado falta sumar la constante de integración C.
Ejemplos de aplicaciones serian las siguientes:

```
(IntegralDefinida (* 3 x) 2 0)
2/3

(IntegralDefinida (+ (^ x 3) (^ x 2)) 1 0)
7/12
```