Lenguajes de programación

Listas en lenguaje SCHEME

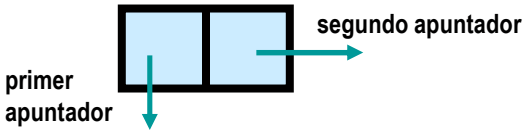
La herramienta "universal" para representar y trabajar con estructuras de datos

Representación en Scheme

- Representación visual:
  - Cero o más **elementos** separados por al menos un espacio y entre paréntesis.
  - Una lista de datos puede contener como **elementos**: números, símbolos o listas de datos.
  - Ej. ( )
    (1 2 3)
    (a b c)
    (lenguajes de programación 10 L)
    (a 1 (b 2) ((c 3) (d 4)) (((e) 5) x))

Representación en Scheme

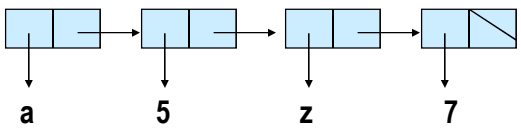
- Representación interna (física):
  - Las listas están almacenadas internamente en memoria dinámica, utilizando encadenamientos de nodos llamados llamados "***celdas cons***".
  - Una ***celda cons*** consiste de dos apuntadores; ambos pueden apuntar a átomos o a otra celda cons.



Representación visual / interna

- La **lista vacía** ***( )*** NO utiliza internamente una *celda cons*, pues es considerada un valor atómico.
- Una lista de 'n' elementos, tendrá internamente 'n' *celdas cons* encadenadas, en donde:
  - la última *celda cons* apunta a la lista vacía con su segundo apuntador, y
  - todos los primeros apuntadores de cada *celda cons* apuntan al elemento correspondiente en la lista.

Ejemplo:
(a 5 z 7)



Operaciones sobre una lista

- *Creación*:
  - Como constantes: utilizando la forma especial *quote*. Ej. *'(1 2 3)*
  - Primitivo de bajo nivel: ***cons***
  - Primitivos de alto nivel: ***list***, ***append***
- *Manipulación de una lista*:
  - Primitivos de bajo nivel: ***car***, ***cdr***
  - Primitivos de alto nivel: ***reverse***, ***list-ref***

Acceso a los elementos de una lista

- Operación ***car***
  - Formato: *(car lista)*
  - Genera como resultado el primer elemento de la lista (valor apuntado por el primer apuntador de la primera *celda cons* de la lista).

    Ejemplo:

    (car '(a b c)) -> a
    (car '((a) (b c) (d) e)) -> (a)
    (car '( )) -> error

Acceso a los elementos de una lista

- Operación ***cdr***
  - Formato: *(cdr lista)*
  - Genera como resultado el "resto" de la lista, es decir, el valor apuntado por el segundo apuntador de la primera *celda cons* de la lista.

    Ejemplo:

    (cdr '(a b c)) -> (b c)
    (cdr '((a) (b c) (d) e)) -> ((b c) (d) e)
    (cdr '( )) -> error

EJERCICIO

- *¿Cómo accesar el segundo elemento de la lista:* ***(x y z)****?*
- ***(car (cdr '(x y z)))***
- ***(cadr '(x y z))***
- *¿Cómo accesar al dato "b" en la lista:* ***(a (b c) d)****?*
- ***(car (car (cdr '(a (b c) d))))***
- *¿Qué resultado se obtiene de:* ***(cddr '(a (b c) d))****?*
- ***(d)***

Construcción de una lista

- Visualmente (listas como constantes):
  - utilizar la forma especial ***quote*** con la lista.
  - Ej. ‘(1 2 3 4 5) ‘(((a 1) (b 2)) (+ 4 x))
- Funcionalmente (listas en ejecución):
  - Procedimiento primitivo de bajo nivel: ***cons***
  - Procedimientos primitivos de alto nivel: ***list*** , ***append***

Construcción de una lista: Operación cons

- Formato: *(cons arg1 arg2)*
- Crea una “*celda cons”* en memoria, en donde el primer apuntador apunta a *arg1* y el segundo a *arg2*.
- Utiliza un orden de evaluación aplicativo, es decir, primero se evalúan los argumentos y después se utilizan los resultados para conformar la *celda cons*.

Tipos de listas

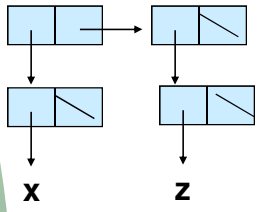
- Lista **PROPIA**
  - es aquella en que el último nodo de la lista apunta (con su segundo apuntador) a la lista vacía.
  - Visualmente utiliza el formato normal.
- Lista **IMPROPIA**
  - es aquella en que el último nodo de la lista apunta (con su segundo apuntador) a un elemento diferente a la lista vacía.
  - Visualmente utiliza un punto entre el primer y segundo elemento de la lista. Ej. (a . b)

Ejemplos : Uso del cons

- *¿Qúe se obtiene de :* (cons 3 ())?
- (3)
- *¿Qué se obtiene de :* (cons 5 8)?
- (5 . 8)
- *¿Cómo se puede construir la lista:* (a b c)?
- *(cons 'a (cons 'b (cons 'c ( ))))*
- *¿Cómo se puede construir la lista:* ((9) 2)?
- *(cons (cons 9 ( )) (cons 2 ( )))*

Ejemplo

- ¿Cuál es la representación visual de la siguiente lista? ¿cómo se construye?



- ((x) (z))
- *(cons (cons 'x ( )) (cons (cons 'z ( )) ( )))*

Construcción de una lista: Operación list

- Formato: *(list arg1 arg2 arg3 ...)*
- Crea una lista propia en la que cada argumento, pasa a ser un elemento de la lista (en el orden correspondiente).
- Utiliza un orden de evaluación aplicativo, es decir, primero se evalúan los argumentos y después se utilizan los resultados para conformar la *celda cons*.
- Ej. *(list 1 2 3)* --> (1 2 3)

Construcción de una lista: Operación append

- Formato: *(append lista1 lista2 …)*
- Genera una lista consistente en los elementos de la primera lista seguidos por los elementos de las otras listas.
- El último argumento puede ser cualquier tipo de objeto, pero si no es una lista propia regresará una lista impropia.
- Ej. *(append '(1 2 3) (cons 'a (list 'b 'c)))*
  --> (1 2 3 a b c)

Recursividad en listas

- **CASO BASE**: Normalmente, esta basado en la solución al caso de la lista vacía.
- **CASO GENERAL**: Normalmente, supone que se tiene la solución al caso del "resto" (*cdr*) de la lista, y utiliza ese resultado para calcular el resultado general.

Predicados relacionados con el manejo de listas

- ***(null? arg)***
  - Verifica si su argumento es la lista vacía o no.
- ***(list? arg)***
  - Verifica si su argumento es una lista **propia**.
- ***(pair? arg)***
  - Verifica si su argumento es una lista que contiene al menos una *celda cons*.

Predicados relacionados con el manejo de listas

- ***(equal? arg1 arg2)***
  - Verifica si sus dos argumentos son idénticos en contenido.
- ***(eq? arg1 arg2)***
  - Verifica si sus dos argumentos son idénticos en contenido y en su representación física.

Ejemplo:

(define L1 (list 1 2 3)) (define L2 (list 1 2 3)) (define L3 L1)

Resultados verdaderos: ***(equal? L1 L2) (eq? L1 L3)***

Resultado falso: ***(eq? L1 L2)***

Otros procedimientos para el manejo de listas

- ***(length lista)***
  - Regresa la longitud de una lista **propia**, o sea, el número de elementos que contiene.
- ***(reverse lista)***
  - Regresa una nueva lista con los elementos de la lista **propia** pasada como argumento en el orden inverso.

Otros procedimientos para el manejo de listas

- ***(list-ref lista k)***
  - Regresa el k-ésimo elemento de una lista (apuntado por el primer apuntador de la k-ésima celda cons)
- ***(list-tail lista k)***
  - Regresa una sublista de lista omitiendo los primeros k elementos.

Tipos de problemas para manejar listas

- Problemas que trabajan con **listas planas**, o sólo con los elementos de su primer nivel.
- Casos de **RECURSIVIDAD PLANA:**
  - Problemas que reciben al menos una lista y generan un resultado atómico.
  - Problemas que reciben solo valores atómicos y generan una lista.
  - Problemas que reciben al menos una lista y generan una lista.

Ejemplo: Lista → átomo

- Contar los elementos de una lista
  (cuenta '(1 2 3 4)) → 4
  (cuenta '(a (b c (d)))) → 2
  (cuenta '()) → 0
- **CASO BASE**: lista vacía
  # elementos = 0
- **CASO GENERAL:** lista con elementos
  # elementos = sumar 1 (*primer elemento*) a la cantidad de elementos restantes

Ejemplo: Átomo → Lista

- Generar una lista con 'n' ceros.
  (ceros 0) → ()
  (ceros 4) → (0 0 0 0)

- **CASO BASE**: n = 0
  Resultado = lista vacía = ( )
- **CASO GENERAL**: n > 0
  Resultado = agregarle un cero (*n-ésimo cero*) a una lista que contenga n-1 ceros

Ejemplo: Lista → Lista

- Incrementar los elementos de una lista.
  (incrementa '()) → ()
  (incrementa '(1 2 3 4)) → (2 3 4 5)
  (incrementa '(3 1 -2 0)) → (4 2 -1 1)

- **CASO BASE**: lista vacía
  Resultado = lista vacía = ( )
- **CASO GENERAL**: lista con elementos
  Resultado = agregar el primer elemento original incrementado a una lista que contiene el resto de los elementos ya incrementados