Ejercicios de tipos de datos recursivos

Programación — DAW

Ricardo Pérez López
IES Doñana

Curso 2020/2021

1. La función `elem` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{elem(}e\texttt{, }t\texttt{ : tuple) -> bool} \\ \textbf{Post}: & \texttt{elem(}e\texttt{, }t\texttt{)} = \begin{cases} \texttt{True} & \text{si } e \text{ está en } t \\ \texttt{False} & \text{en caso contrario} \end{cases} \end{cases}$$

   Escribir una función recursiva que satisfaga dicha especificación.

2. La función `cuantos` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{cuantos(}e\texttt{, }t\texttt{ : tuple) -> int} \\ \textbf{Post}: & \texttt{cuantos(}e\texttt{, }t\texttt{)} = \text{el número de veces que aparece } e \text{ en } t \end{cases}$$

   Escribir una función recursiva que satisfaga dicha especificación y que genere un proceso:

   a) recursivo.

   b) iterativo.

3. La función `quita` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{quita(}e\texttt{, }t\texttt{ : tuple) -> tuple} \\ \textbf{Post}: & \texttt{quita(}e\texttt{, }t\texttt{)} = \text{una tupla igual que } t \text{ pero sin los } e \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación y que genere un proceso:

a) recursivo.

b) iterativo.

4. La función `sustituye` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & \texttt{True} \\ & \texttt{sustituye(}a\texttt{, }b\texttt{, }t\texttt{ : tuple) -> tuple} \\ \textbf{Post}: & \texttt{sustituye(}a\texttt{, }b\texttt{, }t\texttt{)} = \text{una tupla igual que } t \text{ pero} \\ & \text{sustituyendo los } a \text{ por } b \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación y que genere un proceso:

a) recursivo.

b) iterativo.

5. La función `ultimo` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & t \neq \texttt{()} \\ & \texttt{ultimo(}t\texttt{ : tuple)} \\ \textbf{Post}: & \texttt{ultimo(}t\texttt{)} = \text{el último elemento de } t \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación.

6. La función `enesimo` tiene la siguiente especificación:

$$\begin{cases} \textbf{Pre}: & t \neq \texttt{()} \land 0 \leq n < \texttt{len(}t\texttt{)} \\ & \texttt{enesimo(}n\texttt{ : int, }t\texttt{ : tuple)} \\ \textbf{Post}: & \texttt{enesimo(}n\texttt{, }t\texttt{)} = \text{el } n\text{-ésimo elemento de } t \end{cases}$$

Escribir una función recursiva que satisfaga dicha especificación.

Soluciones

1. ```
   elem = lambda e, t: False if t == () else \
                       True if t[0] == e else \
                       elem(e, t[1:])
   ```

2. Definimos:

   ```
   aux = lambda a, b: 1 if a == b else 0
   ```

   a)
   ```
   cuantos = lambda e, t: 0 if t == () else \
                          aux(e, t[0]) + cuantos(e, t[1:])
   ```

   b)
   ```
   cuantos = lambda e, t: cuantos_it(e, t, 0)
   cuantos_it = lambda e, t, acc: acc if t == () else \
                                  cuantos_it(e, t[1:], acc + aux(e, t[0]))
   ```

3. Definimos:

   ```
   aux = lambda a, b: () if a == b else (b,)
   ```

   a)
   ```
   quita = lambda e, t: () if t == () else \
                        aux(e, t[0]) + quita(e, t[1:])
   ```

   b)
   ```
   quita = lambda e, t: quita_it(e, t, ())
   quita_it = lambda e, t, acc: acc if t == () else \
                                quita_it(e, t[1:], acc + aux(e, t[0]))
   ```

4. Definimos:

   ```
   aux = lambda a, b, t: (b,) if a == t else (t,)
   ```

   a)
   ```
   sustituye = lambda a, b, t: () if t == () else \
                               aux(a, b, t[0]) + sustituye(a, b, t[1:])
   ```

   b)
   ```
   sustituye = lambda a, b, t: sustituye_it(a, b, t, ())
   sustituye_it = lambda a, b, t, acc: \
       acc if t == () else \
       sustituye_it(a, b, t[1:], acc + aux(a, b, t[0]))
   ```

5. ```
   ultimo = lambda t: t[0] if t[1:] == () else ultimo(t[1:])
   ```

6. ```
   enesimo = lambda n, t: t[0] if n == 0 else enesimo(n - 1, t[1:])
   ```
   ```