Question 2

Bookmark this page

Question 2

1/1 point (graded)

Here is the `fold-element` function we just designed (check-expects have been omitted to save space):

```
;; (String Integer Y -> X) (X Y -> Y) Y Element -> X
;; the abstract fold function for element
(define (fold-element c1 c2 b e)
  (local [(define (fn-for-element e)
            (c1 (elt-name e)
                (elt-data e)
                (fn-for-loe (elt-subs e))))
          (define (fn-for-loe loe)
            (cond [(empty? loe) b]
                  [else
                   (c2 (fn-for-element (first loe))
                       (fn-for-loe (rest loe)))]))]
    (fn-for-element e)))
```

We want to design a function using `fold-element` that consumes an element and produces the sum of all the data in that element and its subs. Here is the partial function design:

```
;; Element -> Natural
;; produces the sum of all data in element (and its subs)
(check-expect (sum-data F1) 1)
(check-expect (sum-data D5) 3)
(check-expect (sum-data D4) (+ 1 2))
(check-expect (sum-data D6) (+ 1 2 3))

(define (sum-data e) 0)    ;stub
```

What is the correct function body for `sum-data`?

```
(define (sum-data e)
  (fold-element + + 0 e))
```

```
(define (sum-data e)
  (fold-element + + 1 e))
```

```
(define (sum-data e)
  (local [(define (c1 name data loe) (+ data loe))]
    (fold-element c1 + 0 e)))
```

```
(define (sum-data e)
  (local [(define (c1 name data loe) (+ data loe))
          (define (c2 b e loe) (+ e loe))]
    (fold-element c1 + 0 e)))
```

Explanation

We do not need the element name, so we have do design a local function for `c1`. For `c2` we can just add the result of the natural recursion and the result of `fn-for-element (first loe))`. The base case value is 0.

Submit Show Answer

Answers are displayed within the problem