MAC5754
Exercício Programa 2
Data de entrega: 30/11/2021 23:59:59

Instruções:
1 . Este programa será feito em duas partes. Na primeira você irá fazer alguns exercícios de avaliação por demanda utilizando streams do Racket. Na segunda parte você vai desenvolver um interpretador de nossa linguagem com a avaliação por demanda do SASL Você deve entregar seu programa pelo PACA em um arquivo único .rkt contendo as definições do interpretador e das funções da primeira parte.
2. Programas atrasados terão uma penalidade de 20% da pontuação POR DIA.
3. Para a parte 1 você deve usar o dialeto "scheme", pois o plai-typed não tem Streams.
4. Para a segunda parte vocês devem se basear no interpretador com a solução do Ep1, disponível no e-disciplinas.

I. Parte prática (6 pontos)

IMPORTANTE: não será dado crédito parcial aos itens., apenas aqueles que funcionarem receberão pontos. Para esta parte você deve usar *streams* do plai, para que possa usar listas infinitas adiadas.

1. Utilizando o método dos esquemas de recursão dados em classe, defina as seguintes funções
    a. (itera f x) que retorna a ista infinita (x, f(x), f(f(x))....)
    b. (ciclo l) que recebe como parâmetro uma lista finita l e retorna a repetição infinita de l. Assim teremos:
        ->(ciclo '(1 2 3 4))
        (1 2 3 4 1 2 3 4 1 2 3 4 ....)
    c. (fib) que retorna a lista de todos os números de Fibonacci, isto é, a lista (1 1 2 3 5 8...). Ei seus comentários, d&e a Fórmula geral para esquemas de recursão do tipo

x_0 = <valor inicial$_0$>
x_1 = <valor inicial$_1$>
x_2 = <valor inicial$_2$>
$x_{i+3} = f(x_{i+2}, x_{i+1}, x_i, i+3)$

2. Escreva a função (merge l1 l2) que recebe duas listas infinitas (x_1 ....) (y_1 ...) e as junta numa única lista (x_1 ,y_1,x_2 ,y_2,.....)
3. A função *foreach* pode ser definida como:
   (define foreach (lambda (lista f)
   (if (null? lista) '()
   (append (f (car lista)) (foreach (cdr lista) f)))))

   Defina a função *foreach-inf* , similar a *foreach*, com duas diferenças. Primeiro, a lista passada como parâmetro pode ser infinita. Segundo, a função *foreach-inf* deve fazer o "merge" dos resultados de aplicar a função dada aos elementos da lista e não o "append". Ou seja, cada elemento da lista pode ser uma lista infinita.
   Assim se tivéssemos
   (define inteirosde2 (ints-from 2))
   (define inteirosde10 (ints-from 2))
   (foreach-ing (list inteirosde2 inteirosde10) square)
   O resultado seria a lista infinita
   (4 100 9 121 .....)
   Note que o primeiro elemento do resultado é o quadrado do primeiro elemento da primeira lista, o segundo elemento do resultado é o quadrado do primeiro elemento da segunda lista. O terceiro elemento do resultado é o quadrado do segundo elemento da segunda lista e assim por diante)

II. Interpretador (10 pontos)

Nesta parte vamos implementar o interpretador de SASL. Para podermos experimentar exemplos não triviais precisamos aumentar a linguagem.

a. (2 pontos) implemente a função "equal?" que faz a comparação entre dois valores quaisquer (dica: use *equal?* do plai-typed e fica fácil fazer para todos os valores)
b. implemente avaliação por demanda. Neste ano vamos usar uma implementação semelhante àquela do dialeto "lazy"do

racket. Para isso você deve criar dois novos tipo: promise e suspV. O primeiro será usado em células e no ambiente para conter algo que pode ser uma suspensão ou um valor normal. O conteúdo de um *promise* deve ser alterável, ou seja, deve utilizar *boxes.* Quando acessarmos uma promise, devemos verificar se o conteúdo é um suspV ou outro valor. No primeiro caso a suspensão deve ser calculada e substituída no 'promise' pelo valor resultante. No segundo caso o valor é retornado diretamente.

c. Lembrem-se suspV deve ser semelhante a um closV, mas sem o argumento. Além disso é importante que ele seja calculado apenas uma vez.
d. Para facilitar vamos todos usar a mesma definição de promise:

```
(define-type Promise
     [aPromise (valueBox : (boxof  Value))])
```

e. Além disso você deve usar a função auxiliar query-promise para acessar (e, eventualmente, trocar) o valor contido numa *promise:*

```
(define (query-promise [prom : Promise]) : Value
 (let ((theBox (aPromise-valueBox prom)))
  (type-case Value (unbox theBox)
   [susV (body susp-env)
         (let* ((finalValue (interp body susp-env)))
               (begin (set-box!  theBox finalValue)
                     finalValue))]
   [else (unbox theBox)]
   )))
```