Trabalho Paradigma Funcional

Você pode escolher entre duas opções de trabalho. Antes de apresentar as opções eu gostaria de falar um pouco sobre os objetivos e os critérios de avaliação e também dar algumas dicas.

Objetivos e critérios de avaliação

Os objetivos do trabalho são:

- Exercitar a criação de programas utilizando o paradigma de programação funcional
- Exercitar a criação de programas usando um processo sistemático

Considerando esses objetivos, os trabalhos serão avaliados de acordo com os seguintes critérios:

- Utilização do processo sistemático
    - As funções devem ter assinatura, propósito e exemplos
    - O corpo das funções devem ser escritas de acordo com os modelos
- Definições apropriada de tipos de dados
- Uso adequado de composição de funções
- Uso adequado de funções de alta ordem
- Simplicidade e facilidade de entendimento do código
- Funcionamento
    - O programa deve funcionar de acordo com o especificado
    - Os testes devem passar

Dicas

Segue algumas dicas que podem ajudar vocês na implementação do trabalho:

- Siga a receita de projeto e use os modelos adequados, eles vão te ajudar a pensar na solução
- Atenção para o modelo baseado em composição de funções. Se a função deve fazer duas ou mais tarefas e uma tarefa precisa do resultado da outra, então use o modelo de composição de função. Observe se alguma das tarefas pode ser feita com alguma função de alta ordem (veja o próximo item).
- Atenção para o uso de funções de alta ordem. Se uma função se adéqua ao uso de uma função de alta ordem, como `map`, `filter`, `foldr`, `foldl`, `andmap`, `ormap`, etc, use a função de alta ordem!
- Se o código está ficando muito complicado e você está achando que uma solução imperativa ficaria mais simples, reveja com cuidado a sua abordagem. Apesar de ser possível que a solução de um problema seja melhor expresso de forma imperativa

do que funcional, eu acho que esse não é o caso para os problemas que estamos resolvendo.

Opções

Agora as opções de trabalho.

Criação de um Jogo

Aqui você deve criar um jogo usando a receita "Como projetar mundos". Apesar de não termos visto essa receita na disciplina, você pode assistir aos vídeos (listados abaixo) para aprender como projetar programas interativos. Note que a maior parte do trabalho você vai usar o que vimos em sala.

Esta opção é mais adequada para quem fez o curso de jogos, mas qualquer aluno pode escolher essa opção.

Se você fez o curso de jogos e entregou o projeto final, você pode reaproveitar o projeto e entregá-lo como trabalho dessa disciplina, mas você vai ter que fazer alguns ajustes. Você vai ter que usar a linguagem Racket ao invés da BSL. Você vai ter que ajustar o código para usar funções de alta ordem onde for apropriado. Se o seu programa não usa lista, você vai ter que modificá-lo para usar. E é isso! Se você tiver dúvidas de como adaptar o seu programa, entre em contato comigo.

Links

Como projetar mundos (curso de jogos) - Os vídeos são um pouco longos, mas estão em português. Se vocês tiverem problemas para acessar os arquivos, me avisem.
Como projetar mundos (em inglês) - Vídeos curtos e exemplos simples, mas está em inglês.
Como projetar mundos com dados compostos (em inglês)

Frequência de números em uma lista

Projetar uma função que receba como entrada uma lista de strings e um número natural $k > 0$ e produza como saída uma lista com as k strings mais frequentes e o número de vezes que elas aparecem. Você não pode usar nenhuma função de lista da biblioteca do Racket, você tem que escrever todas as funções de lista que você vai utilizar. Você pode utilizar qualquer função que foi definida em aula, para isso você deve informar no código que a função veio do material e apresentar o projeto completo da função. Você não precisa se preocupar com a eficiência da solução, apenas que ela funcione.

Segue alguns comentários de como iniciar o projeto. Vamos precisar definir um tipo para representar a frequência de uma string. Vamos chamar este tipo de `Frequencia` (são dois campos, a string e o número de vezes que ela aparece). Podemos observar que a função tem que fazer duas tarefas distintas, a primeira é calcular a frequência e a segunda é

selecionar as k strings com maior frequência. Então podemos utilizar composição de função e criar o seguinte "esboço" inicial do solução:

```
;; Lista(String) Natural -> Lista(Frequencia)
;; Calcula a frequência das palavras em lst
;; e produz uma lista com as k palavras com
;; maiores frequências.
;; TODO: O que fazer se k é maior que o número
;; de palavras distintas de lst?
(examples
 (check-equal? (palavras-mais-frequentes
                (list "cpu" "a" "casa" "casa" "cpu" "casa")
                2)
               (list (make-frequencia "casa" 3)
                     (make-frequencia "cpu" 2))))

(define (palavras-mais-frequentes lst k)
  (mais-frequentes k (calcula-frequencia lst)))
```

E adicionamos as duas funções na lista de desejos:

```
;; Lista(String) -> Lista(Frequencia)
;; Calcula a frequência das palavras em lst
(define (calcula-frequencia lst) empty)

;; Natural Lista(Frequencia) -> Lista(Frequencia)
;; Produz uma lista com os k elementos de
;; lst que tem maior frequência.
(define (mais-frequentes k lst) lst)
```