LGP: Linguagens de Programação / 2020
Atividade 03

Introdução (IMPORTANTE!)

Nessa lista de exercícios, nós vamos trabalhar com arquivos JSON em SML. Nós vamos fazer a manipulação desses arquivos, imprimir esses arquivos, e (opcional, como um problema desafio) fazer o *parsing* desses arquivos.
Você precisa fazer o download do arquivo anexo ao enunciado. Você vai editar o arquivo hw2.sml. As soluções devem usar *pattern-matching*. Você não deve usar as funções **`null, hd, tl, isSome`**, ou **`valOf`**, nem deve usar qualquer coisa que contenha o caractere **`#`** ou funcionalidades que não foram estudadas nas notas de aula da Unidade 2 (por exemplo, mutação). Observe que a ordem dos exercícios não importa, a menos que seja especificado no enunciado.
Esta lista de exercícios é um pouco mais trabalhosa que a anterior. Temos 8 exercícios obrigatórios e nenhum deles precisa de mais de 25 linhas de código para resolver.

Visão Geral

JSON é um formato de dados serializados legível para nós humanos, originalmente inspirado pela notação de objeto do Javascript. JSON é comumente usado para representar dados.
Não se preocupem! Não é necessário nenhum conhecimento prévio sobre manipulação de arquivos JSON. Aqui não seremos muito rigorosos ao tratar desse formato de arquivo, então não será muito difícil.
Um valor JSON é uma das seguintes coisas:

1. Um número (ponto flutuante, por exemplo 3.14 ou -0.2)
2. Uma string (por exemplo, "Ola!")
3. **`False`**
4. **`True`**
5. **`Null`**
6. Um array de valores JSON, escrito usando colchetes e vírgulas (por exemplo, **`[1, "mundo", null]`**)
7. Um "objeto", que é uma sequência de pares atributo/valor
    a. Um atributo é sempre uma string literal (por exemplo "**`foo`**")
    b. Um valor é qualquer valor JSON
    c. Objetos são escritos usando chaves, vírgulas, e dois pontos. Por exemplo: **`{"foo" : 3.14, "bar" : [1, "world", null], "ok" : true}`**

No entanto, nosso Código ML não usa essa notação JSON, ao invés disso usamos *datatypes* para representar os valores JSON:

```
datatype json =
       Num of string
       | String of string
       | False
       | True
       | Null
       | Array of json list
       | Object of (string * json) list
```

Lembre-se que para qualquer exercício, se um tipo é for pedido, então um tipo mais geral é aceitável dado que seu código esteja correto.

Exercícios

1. Escreva uma função **`criar_json`** que recebe um inteiro **`u`** e retorna um JSON. O resultado deve representar um *array* JSON de objetos JSON em que todo objeto nesse *array* tem dois atributos: **`"n"`** e **`"b"`**. Todo atributo **`"b"`** de um objeto deve ser **`true`**. O primeiro objeto no *array* deve conter um atributo **`"n"`** que guarda o número JSON **`i.0`**, o próximo objeto deve ter seu atributo **`"n"`** guardando **`(i-1).0`** e assim por diante até que o último objeto no *array* tenha seu atributo **`"n"`** guardando o valor **`1.0`**. Se **`i`** for **`0`**, deve ser produzido um *array* com **`0`** objetos. Assuma que i nunca será negativo. É possível resolver esse exercício com menos de 10 linhas. *Dica: Foi disponibilizada uma função* **`int-to-real`**. Você vai precisar também de uma função auxiliar que faz quase que todo o trabalho necessário.

2. Escreva uma função **`assoc`** do tipo **`"a * ("a * 'b) list -> 'b option`** que recebe dois argumentos **`k`** e **`xs`**. Essa função deve retornar **`SOME v1`** se **`(k1, v1)`** é o par da lista que está mais próximo ao começo da lista para a qual **`k`** e **`k1`** são iguais. Se não existir um par **`(k1, v1)`** ideal, a função deve retornar **`NONE`**. A solução também requer poucas linhas de código.

3. Escreva uma função **`dot`** que recebe um JSON (chame de **`j`**) e a uma *string* (chame de **`f`**) e retorna uma JSON **`option`**. Se **`j`** for um objeto que tem um atributo chamado **`f`**, então a função deve retornar **`SOME v`**, onde **`v`** é o conteúdo do atributo. Se **`j`** não é um objeto ou não contém um atributo **`f`**, então a função deve retornar **`NONE`**. É possível resolver esse exercício com 4 linhas de código e usando um dos exercícios anteriores.

4. Escreva uma função **`um_atributo`** que recebe um JSON e retorna uma **`string list`**. Se o argumento é um objeto, a função deve retornar uma lista contendo todos os nomes dos atributos (mas não o conteúdo dos atributos). Caso contrário, uma lista vazia deve ser retornada. Use uma função auxiliar local com recursão de calda. Pode ser que apareça como resultado uma *string* em ordem reversa daquela que aparece no objeto, se isso acontecer está correto também. A solução precisa de aproximadamente 11 linhas de código.

5. Escreva uma função **`sem_repeticao`** que recebe uma **`string list`** e retorna um **`bool`** que é **`true`** se e somente se a *string* aparece mais de uma vez na entrada recebida. Não **(!)** use nenhuma recursão explícita. Ao invés disso, use as funções auxiliares **`lenght`** (pré-definida na SML) e **`dedup`** (disponibilizada em arquivo) para resolver esse exercício usando só uma linha de código.

6. Escreva uma função **`sem_atributos_repetidos_recursivos`** que recebe um JSON e retorna um **`bool`** que é **`true`** somente se nenhum objeto "dentro" (arbitrariamente aninhado) do argumento JSON tiver nomes repetidos para atributos.
7. Escreva uma função **`contar_ocorrencias`** do tipo **`string list * exn -> (string * int) list`**. Se o argumento **`string list`** estiver ordenado (em termos da ordenação da função **`strcmp`** que foi fornecida para vocês), a função deve retornar uma lista em que cada string é pareado com o número de vezes que ela ocorre. (A ordem na lista resultante não importa). Se a lista não estiver ordenada, lançar o argumento **`exn`**. Sua implementação deve fazer uma única passada sobre a lista **`string list`**, usando uma função auxiliar com recursão de calda. A função auxiliar pode precisar de alguns argumentos, que incluem a string "atual" e o contador de "atual". É possível solucionar em aproximadamente 14 linhas.

8. Escreva uma função **`valores_string_para_atributos`** do tipo **`string * (json list)-> string list`** (os parênteses nesse tipo são opcionais, o REPL não irá imprimi-los). Para qualquer objeto na **`json list`** que tenha um atributo igual a **`string`** e que tenha conteúdo que seja uma string json (por exemplo, **`String "hi"`**) coloque o conteúdo da string (por exemplo **`"hi"`**) na lista resultante (a ordem não importa, a saída pode ter repetições quando necessário). Assuma que nenhum objeto tenha nomes repetidos para atributos. É possível solucionar com 6 linhas graças à função **`dot`**.