UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
LISTA OBRIGATÓRIA DE PROGRAMAÇÃO FUNCIONAL – PARTE 1

Professores: Giovanny Fernando Lucero Palma e Leila Maciel de Almeida e Silva

Esta lista deve ser feita usando **apenas compreensões** e as **funções básicas sobre listas e tuplas.** Nesta atividade você deve implementar todas as funções previstas para o funcionamento do sistema descrito a seguir, bem como funções auxiliares que julgue necessárias.

Considere um sistema simplificado que controla a vacinação de COVID. O sistema exige que todos os cidadãos se registrem no SUS para realizarem a vacinação. O cidadão pode realizar seu cadastro no SUS via internet, a fim de não sobrecarregar o atendimento dos postos de saúde. O arquivo que armazena os dados do cadastro corrente do SUS tem a seguinte declaração de tipo:

```
type CadastroSUS = [Cidadao]
```

Cada item desse cadastro, `Cidadao`, é da forma

```
type CPF = Integer
type Nome = String
type Genero = Char
type Dia = Int
type Mes = Int
type Ano = Int
type Data = (Dia, Mes, Ano)
type DataNasc = Data
type Endereco = String
type Municipio = String
type Estado = String
type Telefone = String
type Email = String
type Cidadao = (CPF, Nome, Genero, DataNasc, Endereco, Municipio,
                  Estado, Telefone, Email)
```

Um exemplo de um arquivo de cadastro com dados do SUS poderia ser:

```
meuCadastro :: CadastroSUS
meuCadastro =
  [(26716347665, "Paulo Souza", 'M', (11,10,1996),"Rua A, 202",
      "Muribeca", "SE", "999997000", "psouza@gmail.com"),
   (87717347115, "Ana Reis",'F', (5,4,1970), "Rua B, 304",
      "Aracaju", "SE", "999826004", "areis@gmail.com")]
```

O CPF será a chave para localizar os dados de um cidadão neste cadastro, por ser único para cada cidadão. Para simplificar, considere apenas três tipos de gênero, feminino (F),

masculino (M) e demais tipos (X). Para manipular este cadastro, algumas funções precisam ser implementadas.

(a) Cadastramento de um cidadão no sistema.

```
adicionaSUS :: Cidadao -> CadastroSUS -> CadastroSUS
```

Para cadastrar um novo cidadão, inicialmente é checado se o CPF já existe ou não no sistema com a função

```
checaCPF :: CPF -> CadastroSUS -> Bool
```

Se o CPF for inexistente, os dados do cidadão serão adicionados ao cadastro corrente. Nesta atividade, você informará todos os dados que compõem a tupla do tipo `Cidadao`. Posteriormente, aprenderemos como coletar os dados da tupla a partir da interação com o usuário, usando o teclado. Se o CPF já existir, uma mensagem de erro, usando error, deve ser exibida informando que o usuário já está cadastrado.

(b) O cidadão pode querer modificar algum desses dados, por exemplo, o número de telefone ou endereço. Para isto, precisamos de funções de atualização dos dados no cadastro, passando os novos dados. Para simplificar o sistema, vamos supor apenas as funções de atualização do endereço e do telefone, já que as demais atualizações seguiriam o mesmo princípio. No processo de atualização, o cadastro SUS informado será copiado para um novo cadastro SUS. Neste novo cadastro, os registros de outros cidadãos permanecerão inalterados e somente os dados do cidadão que está sendo atualizado sofrerão modificações.

```
atualizaEndSUS :: CPF -> CadastroSUS -> Endereco -> CadastroSUS
atualizaTelSUS :: CPF -> CadastroSUS -> Telefone -> CadastroSUS
```

(c) Quando um cidadão falece, a família tem que notificar o fato em um posto de saúde, para que ele seja retirado do cadastro corrente do SUS. Como há uma verificação do atestado de óbito, isto só pode ser feito no posto. O sistema precisará da função abaixo. Se o CPF existir no cadastro corrente do SUS, o registro do cidadão deve ser completamente excluído, gerando um novo cadastro sem os dados deste cidadão. Se o CPF não existir, uma mensagem de erro, usando error, sinalizando que o cidadão não pertence ao cadastro deve ser exibida.

```
removeSUS :: CPF -> CadastroSUS -> CadastroSUS
```

(d) Um gestor de saúde pode querer pesquisar algumas informações deste cadastro, como por exemplo, quantidade de cidadãos por município, por estado, ou ainda por município e por faixa de idade, ou por estado e por faixa de idade, para ter uma ideia de como planejar as faixas de vacinação. Assim, o sistema deve prever algumas funções de consulta:

```
type IdadeInicial = Int
type IdadeFinal = Int
type FaixaIdade = (IdadeInicial, IdadeFinal)
type Quantidade = Int
cidadaosPorMunicipio :: CadastroSUS -> Municipio -> Quantidade
cidadaosPorEstado :: CadastroSUS -> Estado -> Quantidade
cidadaosPorMunicipioIdade :: CadastroSUS -> Municipio-> FaixaIdade ->
                                    Quantidade
cidadaosPorEstadoIdade :: CadastroSUS -> Estado -> FaixaIdade ->
                                    Quantidade
```

(e) Pode ser interessante também gerar uma lista da quantidade de cidadãos por faixas de idade para um dado município ou estado. As faixas de idade inicialmente previstas são:

FAIXAS DE IDADE

81 - 130
71 - 80
61 - 70
51 - 60
41 - 50
31 - 40
21 - 30
12 - 21

O gestor pode escolher todas ou algumas destas faixas para gerar a lista. O gestor pode também, a depender das características de seu município, escolher outras faixas, já que a faixa é um parâmetro da função. Caso o gestor decida, por exemplo, coletar dados para uma idade específica, digamos 25 anos, ele deve informar a faixa (25, 25).

```
listaMunicipioFaixas :: CadastroSUS -> Municipio -> [FaixaIdade] -> IO()
listaEstadoFaixas :: CadastroSUS -> Estado-> [FaixaIdade] -> IO()
```

Estas funções precisam de funções auxiliares para gerar uma lista de quantidades por faixas de idade, para depois gerar a exibição usando as funções do item (f).

```
geraListaMunicipioFaixas :: CadastroSUS -> Municipio -> [FaixaIdade] ->
                                [(FaixaIdade, Quantidade)]
geraListaEstadoFaixas :: CadastroSUS -> Estado -> [FaixaIdade] ->
                                [(FaixaIdade, Quantidade)]
```

(f) A lista do município ou estado deve obedecer à formatação, descrita a seguir. O cabeçalho segue o formato abaixo, onde nome é o nome do município ou estado informado na função.

MUNICIPIO: nome

ou

ESTADO: nome

Após o cabeçalho, pulando uma linha, deve vir a seguinte tabela, onde a primeira coluna está justificada à esquerda e a segunda coluna, à direita. Entre as palavras IDADE e QUANTIDADE devem ser previstos 15 espaços em banco. Na tabela abaixo, os valores apresentados para a quantidade são ilustrativos.

FAIXAS DE IDADE	QUANTIDADE
81 - 130	900
71 - 80	9000
61 - 70	60380
51 - 60	100200
41 - 50	200030
31 - 40	500008
21 - 30	900200
11 - 20	1001090
TOTAL	2321808

Para atender esta formatação, algumas funções são necessárias:

1. Formatação do valor inteiro que representa a quantidade para incluir os espaços à esquerda, para que a justificação à direita com a palavra QUANTIDADE ocorra.

```
type QuantidadeFormatada = String
formataQuant :: Quantidade -> QuantidadeFormatada
```

   Por exemplo,
   `formataQuant 10000` retornará " 10000"

2. Formatação de uma linha da faixa de idade. A saída desta função será uma string com o formato de uma linha da tabela anterior.

```
type LinhaFormatada = String
formataUmaLinha :: (FaixaIdade, Quantidade)-> LinhaFormatada
```

   Ex: `formataUmaLinha ((71,80), 9000)` retornará

   "71 - 80 9000"

3. Formatação de todas as linhas das faixas de idade. Esta função usará a função anterior, acrescentará `\n` ao final de cada linha formatada e concatenará todas as linhas, gerando uma única string.

```
type LinhasFormatadas = String
formataLinhas :: [(FaixaIdade, Quantidade)] -> LinhasFormatadas
```

4. Formatação da linha de totalização

```
type TotalFormatado = String
formataTotal :: [(FaixaIdade,Quantidade)] -> TotalFormatado
```

Além do cadastro no sistema SUS, haverá um cadastro dos cidadãos vacinados. Este cadastro está definido a seguir:

```
type Vacinados = [Vacinado]
```

Cada item desse cadastro, `Vacinado`, é da forma

```
type Vacina = String
type TipoDose = Int
type Dose = (Vacina, Data)
type Doses = [Dose]
type Vacinado = (CPF, Doses)
```

Para efetuar a vacinação o sistema necessita das funções a seguir. Para simplificar, você pode supor uma lista constante de vacinas possíveis, que são: AstraZeneca, CoronaVac, Janssen e Pfizer. O número das doses segue a posição das tuplas na lista `Doses`. Algumas destas funções requerem a decomposição do problema em funções auxiliares e cabe a você identificar e projetar quais seriam essas funções.

(g) Aplicação da primeira dose.

```
aplicaPrimDose:: CPF -> CadastroSUS -> FaixaIdade -> Municipio ->
                    Vacina -> Data -> Vacinados -> Vacinados
```

Para realizar esta aplicação, procede-se da forma descrita a seguir e algumas funções auxiliares são necessárias. Inicialmente é verificado se o cidadão já tomou uma dose de vacina. Em caso afirmativo, usando error, exibe uma mensagem de que a primeira dose já foi aplicada. Caso contrário, é verificado se o usuário está cadastrado no sistema SUS. Se não estiver cadastrado, exibe uma mensagem de erro sinalizando o problema. Se estiver, checa se a idade é consistente com a faixa de idade de vacinação corrente. Se não for, exibe uma mensagem de erro sinalizando o problema. Se for, checa se o município é coerente com o município do cadastro SUS. Se não for, exibe uma mensagem de erro para ele atualizar os dados do SUS, pois só é permitida a vacinação para residentes no município. Se for, adiciona o usuário no cadastro de vacinados. No momento da adição serão informados os dados constantes em `Vacinado`. Quando a vacina for Janssen, a tupla `Dose` deve vir duplicada na lista `Doses`, sinalizando que o paciente foi completamente imunizado.

(h) Para realizar a segunda dose, faz-se necessário checar se o CPF consta do cadastro de vacinados. Se não estiver, usando error, exibe uma mensagem de erro reportando o problema. Se estiver, é verificado se o cidadão já tomou a segunda dose. Se já

tomou, exibe uma mensagem de erro. Caso contrário, checa se a data informada é maior que a da primeira dose. Se não for, exibe uma mensagem de erro. Se for, o cadastro de vacinados é atualizado e um novo cadastro é gerado, inserindo-se a tupla `Dose` no final da lista `Doses`, para um dado cidadão. Os dados dos demais cidadãos permanecem inalterados.

```
aplicaSegDose:: CPF -> Data -> Vacinados -> Vacinados
```

(i) Pode ser necessária alguma atualização no cadastro de vacinados. Por exemplo, pode ser necessário consertar o nome da vacina que foi informada incorretamente. Para isso, seria necessária a função abaixo onde o nome correto da vacina é informado, para um dado CPF, numa dada dose. Caso o CPF não conste do cadastro de vacinados, exibe uma mensagem de erro. Caso contrário, se o número da dose informado for superior ao tamanho da lista `Doses`, uma mensagem de erro deve ser exibida sinalizando que aquela dose ainda não foi ministrada para aquele cidadão.

```
atualizaVacina:: CPF -> TipoDose -> Vacina -> Vacinados -> Vacinados
```

Algumas informações podem ser úteis para o gestor municipal ou estadual, como:

(j) Quantidade de pessoas no município/estado vacinadas com uma dada dose. Para isso, para cada cidadão no cadastro de vacinados, é verificado se ele já tomou a dose informada no argumento da função. Em caso afirmativo, verifica-se se ele pertence ao município/estado informado, acessando-se o cadastro do SUS, e em caso afirmativo, o cidadão é considerado para o cômputo.

```
QuantidadeDoseMun :: Vacinados -> TipoDose -> Municipio ->
                              CadastroSUS -> Quantidade
QuantidadeDoseEst :: Vacinados -> TipoDose -> Estado ->
                              CadastroSUS -> Quantidade
```

(k) Quantidade de pessoas no município/estado vacinadas por faixa etária e por dose. Procede-se como nos itens anteriores, mas agora se checa, além da dose e do município, a faixa de idade.

```
QuantidadeMunIdDose :: Vacinados -> Municipio -> FaixaIdade ->
                              TipoDose -> CadastroSUS -> Quantidade
QuantidadeEstIdDose :: Vacinados -> Estado -> FaixaIdade ->
                              TipoDose -> CadastroSUS -> Quantidade
```

(l) Quantidade de pessoas no município/estado vacinadas por tipo de vacina e por dose

```
QuantidadeMunVacDose :: Vacinados -> Municipio -> Vacina -> TipoDose ->
                              CadastroSUS -> Quantidade
QuantidadeEstVacDose :: Vacinados -> Estado -> Vacina -> TipoDose ->
                              CadastroSUS -> Quantidade
```

(m) Quantidade de pessoas atrasadas na segunda dose no município/estado, dentre os cidadãos que pertencem ao cadastro de vacinados. Considere que a segunda dose da

CoronaVac deve ser aplicada 21 dias após a primeira dose e a da Pfizer e AstraZeneca 90 dias após a primeira dose.

```
QuantidadeMunAtrasados :: Vacinados -> CadastroSUS -> Municipio ->
                              Quantidade
QuantidadeEstAtrasados :: Vacinados -> CadastroSUS -> Estado ->
                               Quantidade
```

(n) Considerando os dados do cadastro SUS e do cadastro de vacinados elabore e projete duas outras consultas que podem ser feitas sobre esses dados.

(o) Escolha uma das consultas propostas por você no item anterior e idealize como você exibiria um relatório com os dados dessa consulta, com uma formatação adequada, como exercitado no item (f). Projete a função que realiza a exibição do seu relatório.

Para testar o seu programa, defina um cadastro SUS e um cadastro de vacinados. Defina também vários registros para os dois cadastros. Por exemplo, para o cadastro SUS,

```
maria :: Cidadao
maria = (53471688765, "Maria Silva", 'F', (21,12,1984),"Rua A,
            202","Gloria", "SE", "999880300", "msilva@gmail.com")

marcos :: Cidadao
marcos = (53499988765, "Marcos Santos", 'M', (25,10,1994),"Rua D,
            202","Aracaju", "SE", "999880501","msilva@gmail.com")
```

Use esses dados para testar as suas funções ou sequências de funções, valendo-se de `it` para reutilizar a saída de uma função na outra. Por exemplo,

```
adicionaSUS marcos meuCadastro
adicionaSUS maria it
removeSUS 53499988765 it
removeSUS 87717347115 it
```