Primeira Prova de Linguagens de Programação
- DCC024W -
Sistemas de Informação

Nome: _____
"Eu dou minha palavra de honra que não trapacearei neste exame."

Número de matrícula: _____

As regras do jogo:

- A prova é sem consulta.
- Quando terminar, não entregue nada além do caderno de provas para o instrutor.
- Quando escrever código, a sintaxe correta é importante.
- Cada estudante tem direito a fazer uma pergunta ao instrutor durante a prova. Traga o caderno de provas quando vier à mesa do instrutor.
- A prova termina uma hora e quarenta minutos após seu início.
- Seja honesto e lembre-se: **você deu sua palavra de honra**.

Alguns conselhos:

- Escreva sempre algo nas questões, a fim de ganhar algum crédito parcial.
- Se não entender a questão, e já tiver gasto sua pergunta, escreva a sua interpretação da questão junto à resposta.
- A prova não é difícil, ela é divertida, então aproveite!

Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Questão 4	Questão 5	Questão 6

1. (10 Pontos) Considere a gramática abaixo, que está implementada em Prolog. Essa gramática descreve expressões condicionais na linguagem SML. Nesse exemplo simplificado, estamos assumindo que os únicos valores que um programa pode retornar são 0 ou 1:

```
ifexp --> [if], cond, [then], ifexp, [else], ifexp.
ifexp --> [0].
ifexp --> [1].
cond  --> [true].
cond  --> [false].
```

Modifique essa gramática, adicionando um atributo `D` a ela, para que ela seja capaz de contar o número de zeros (0) em uma expressão condicional. Por exemplo:

```
| ?- ifexp(D, [if, true, then, if, false, then, 1, else, 1, else, 0], []).

D = 1

| ?- ifexp(D, [if, true, then, if, false, then, 0, else, 1, else, 0], []).

D = 2

| ?- ifexp(D, [if, true, then, if, false, then, 0, else, 0, else, 0], []).

D = 3
```

2. (10 Pontos) Escreva uma função `toDig`, de tipo `string -> string list`, que converta uma *string* em uma lista de dígitos. Por exemplo:

```
- toDig "123";
val it = ["um","dois","tres"] : string list
- toDig "";
val it = [] : string list
- toDig "001001";
val it = ["zero","zero","um","zero","zero","um"] : string list
```

Essa função deve ser capaz de lidar com qualquer *string* de entrada. Se ela receber uma *string* contendo caracteres que não são números, então ela deve imprimir `???` para cada um desses caracteres. Por exemplo:

```
- toDig "A23";
val it = ["???","dois","tres"] : string list
```

É possível que você queira usar a função `ord`, de tipo `char -> int` em sua solução. Essa função converte um caracter para o seu código ASCII. Por exemplo:

```
- ord #"0";
val it = 48 : int
- ord #"1";
val it = 49 : int
- ord #"2";
val it = 50 : int
```

Outra função que você pode achar interessante é `explode`, de tipo `string -> char list`, que recebe uma *string* e a converte para uma lista de caracteres:

```
- explode "dcc024";
val it = [#"d",#"c",#"c",#"0",#"2",#"4"] : char list
```

3. Uma linguagem de programação L_1 é dita mais dinâmica que outra linguagem de programação L_2 se programas escritos em L_1 *realizam* (ou, doutro modo, *sofrem*) mais testes em tempo de execução.

   - (5 pontos) Qual linguagem é mais dinâmica, Java ou C++? Justifique a sua resposta explicando quais testes são realizados por uma linguagem, e não o são pela outra.

   - (5 pontos) Qual linguagem é mais dinâmica, SML ou Python? Novamente, justifique a sua resposta, explicando quais testes são realizados, em tempo de execução, por uma linguagem, e não pela outra.

4. (10 Pontos) Quase todas as linguagens de programação moderna possuem *escopo estático*, o seja, o escopo de um nome é definido durante a compilação. O outro tipo de escopo, chamado *escopo dinâmico*, está presente na primeira versão da linguagem *Lisp*, e em *APL*, a linguagem inventada por Kenneth Iverson para manipular arranjos. Por que o escopo dinâmico é hoje tão pouco utilizado? Ilustre seus argumentos com código sintaticamente válido em alguma linguagem de programação.

5. (10 Pontos) Considere o tipo algébrico `option` e a função `findName`, de tipo `string -> int option`. O tipo `option` é definido logo abaixo.

```
- datatype 'a option = NONE | SOME of 'a
```

A implementação de `findName` não é importante para essa questão. Considere, contudo, que essa função procure por um dado nome em um banco de dados, e retorne o CPF associado a esse nome, ou `NONE` quando o nome não for encontrado. Por exemplo:

```
- findName "Eliseu LaMarca Passos";
val it = SOME 454886429 : int option
- findName "Augusto D Andrade";
val it = NONE : int option
```

Nessa questão você deve escrever uma função `cpfToString`, cujo tipo seja `string -> string`. Essa função lê um nome, procura pelo mesmo usando `findName` e retorna a *string* `"CPF nao encontrado"`, ou `"CPF = xxxxxxxxx"`, sendo `xxxxxxxxx` o CPF relacionado ao nome dado. Por exemplo:

```
- cpfToString "Eliseu LaMarca Passos";
val it = "CPF = 454886429" : string
- cpfToString "Augusto D Andrade";
val it = "CPF nao encontrado" : string
```

É possível que você queira lançar mão da função `Int.toString`, de tipo `int -> string`, que converte inteiros para *strings*. Por exemplo:

```
- Int.toString 11;
val it = "11" : string
```

6. Nós podemos representar números inteiros usando o cálculo λ. Uma das convenções mais comuns é assumir que um número n é uma função que recebe dois argumentos, e aplica o primeiro ao segundo n vezes. Por exemplo:

   - $0 = \lambda s.\lambda z.z$
   - $1 = \lambda s.\lambda z.sz$
   - $2 = \lambda s.\lambda z.s(sz)$

   (a) (5 pontos) Considere a função sucessor, SUCC $= \lambda n.\lambda y.\lambda x.y(n\ y\ x)$. Essa função recebe uma função descrevendo um número n, e retorna a função correspondente a $n + 1$. Mostre os passos de redução de SUCC 1, ou, doutro modo: $(\lambda n.\lambda y.\lambda x.y(n\ y\ x))(\lambda s.\lambda z.sz)$.

   (b) (5 pontos) Usando a função sucessor, SUCC $= \lambda n.\lambda y.\lambda x.y(n\ y\ x)$, defina a função ADD, que soma dois números.