Primeira Prova de Linguagens de Programação
- DCC024B -
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:

1. Escreva sempre algo nas questões, a fim de ganhar algum crédito parcial.
2. Se não entender a questão, e já tiver gasto sua pergunta, escreva a sua interpretação da questão junto à resposta.
3. 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 extra (0.5): cite algo historicamente importante feito por Júlio César, cidadão romano que viveu entre os anos -100 e -44 da Era Cristã.

1. Seja `insert` uma função que receba um inteiro e uma lista, e retorne uma lista. Essa função possui a seguinte propriedade: caso a lista de entrada esteja ordenada, então a lista de saída está ordenada. O restante desta questão refere-se à `insert`.

   (a) (4 Pontos) Implemente a função `insert`, em SML. O tipo da função `insert` é: `int * int list -> int list`. Abaixo vêem-se alguns exemplos de uso dessa função:

   ```
   - insert (3, [1, 2, 4, 5]);
   val it = [1,2,3,4,5] : int list
   - insert (3, []);
   val it = [3] : int list
   - insert (3, [4, 2, 1]);
   val it = [3,4,2,1] : int list
   ```

   Se a lista de entrada `L` não estiver ordenada, sua implementação não precisa retornar uma lista ordenada.

   (b) (3 Pontos) Caso sucessivas composições da função `insert` sobre inteiros sempre produza listas ordenadas, então temos um algoritmo de ordenação, conforme o exemplo abaixo, que ordena quatro números inteiros:

   ```
   - insert(3, insert(1, insert(2, insert(4, nil))));
   val it = [1,2,3,4] : int list
   ```

   Qual a complexidade desse algoritmo de ordenação, assumindo-se que a função `insert` possui a implementação feita na questão (a)?

   (c) (3 Pontos) Implemente um algoritmo de ordenação usando a função `insert`. Considere a observação feita na questão anterior. Seu algoritmo deve usar uma das funções de redução (`foldr` ou `foldl`) e deve poder ser escrito em uma linha de não mais que quarenta caracteres.

2. Essa questão diz respeito à função `map`, que é parte da biblioteca SML padrão. Alguns exemplos de uso dessa função podem ser vistos logo abaixo:

```
- map (fn x => x * x) [2, 3, 4] ;
val it = [4,9,16] : int list
- map (fn x => Int.toString x ^ "!") [2, 3, 4] ;
val it = ["2!","3!","4!"] : string list
- map (fn x => x > 3) [2, 3, 4] ;
val it = [false,false,true] : bool list
```

(a) (3 Pontos) Qual o tipo da função `map`?

(b) (3 Pontos) Implemente uma função `hds`, que receba uma lista de listas, e retorne uma lista com as cabeças. Por exemplo:

```
- hds [[1, 2, 3], [2, 3, 4], [5]] ;
val it = [1,2,5] : int list
- hds [[1.0, 2.0], [2.0, 3.0, 4.0], [5.0]] ;
val it = [1.0,2.0,5.0] : real list
- hds [["oi", "ei"], ["Fun", "Hung", "Lam", "Jing", "Ling"]] ;
val it = ["oi","Fun"] : string list
```

(c) (4 Pontos) Existe alguma função f em SML, que não pode ser passada para a função `map`? Em outras palavras, sua função f deve gerar um erro de compilação caso tente-se passá-la para a função `apply` abaixo:

```
fun apply f L = map f L ;
```

Caso exista uma função f, você deve mostrar a sua implementação. Caso não exista, você precisa construir um argumento que justifique a sua resposta.

3. Considere a gramática abaixo, que descreve listas em SML:

$$\begin{array}{lcl} \langle L \rangle & ::= & \texttt{nil} \\ & | & \langle L \rangle \ :: \ \langle L \rangle \\ & | & e \end{array}$$

Acima, "e" representa um elemento de uma lista. O terminal `nil` representa a lista vazia.

(a) (3 Pontos) Prove que essa gramática é ambígua.

(b) (3 Pontos) Qual o impacto que a ambiguidade da gramática acima tem sobre a associatividade do operador de construção de listas :: ?

(c) (4 Pontos) Re-escreva a gramática acima, de modo que ela não seja mais ambígua.

4. Esta questão diz respeito à função `halve`, cuja implementação em SML pode ser vista logo abaixo:

```
fun halve nil = (nil, nil)
  | halve [a] = ([a], nil)
  | halve (a::b::cs) =
    let
      val (x, y) = halve cs
    in
      (a::x, b::y)
    end;
```

(a) (2 Pontos) Qual é o tipo da função `halve`?

(b) (2 Pontos) A implementação de `halve` produz uma função polimórfica. Que tipo de polimorfismo é esse, dentre os quatro tipos que existem?

(c) (2 Pontos) Qual o resultado da operação `halve [1, 2, 3, 4, 5]`?

(d) (4 Pontos) Prove a seguinte afirmativa: "Se `halve`$(L) = (L_1, L_2)$, então a diferença de tamanho entre L_1 e L_2 é de no máximo um elemento."

5. Nesta questão você deverá definir alguns termos. Suas definições devem ser escritas em português padrão, mas você precisa ser o mais formal possível. Como um exemplo, segue abaixo a definição de números naturais:

   - *Zero* é um número natural;
   - Se n é um número natural, então o sucessor de n é um número natural.

   Todos os termos externos à sua definição precisam ser sublinhados. Um termo externo à definição é um nome que não vem antes de um verbo de ligação e não é o próprio nome sendo definido.

   (a) (4 Pontos) Defina uma lista.

   (b) (3 Pontos) Defina uma árvore binária.

   (c) (3 Pontos) Defina um grafo.