Primeira Prova de Linguagens de Programação
- DCC024B -
Ciência da Computação

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 resposta para a pergunta "*Posso ir ao banheiro*?" é **sim** (deixe o telefone celular sobre a mesa).
- 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	Extra

Questão Extra (0.5 Pontos): "*Sou brasileiro de estatura mediana; gosto muito de fulana mas ...*":

```
sicrana é quem me quer.
```

1. A questão abaixo envolve a construção de funções lambda em SML/NJ. Podemos definir booleanos no cálculo lambda usando a codificação de Church:

```
val T = fn x => fn y => x
val F = fn x => fn y => y
```

   (a) (2 Pontos) Escreva uma função `print` em SML/NJ, que transforme um booleano de Church em um booleano de SML/NJ:

```
- print T;
val it = true : bool
- print F;
val it = false : bool
```

```
fun print b = b true false
```

   (b) (2 Pontos) Qual o tipo da função `print` que você escreveu acima?

```
(bool -> bool -> 'a) -> 'a
```

   (c) (2 Pontos) Escreva uma função `AND` que produza a conjunção lógica de booleanos de Church:

```
- print (AND T F);
val it = false : bool
- print (AND T T);
val it = true : bool
```

```
val AND = fn b0 => fn b1 => b0 b1 F
```

   (d) (2 Pontos) O tipo da função `AND` deve ser `('a -> ('b -> 'c -> 'c) -> 'd) -> 'a -> 'd`. Este construtor de tipos possui quantos parâmetros?

```
Quatro
```

   (e) (2 Pontos) Escreva uma função `OR` que produza a disjunção lógica de booleanos de Church:

```
- print (OR T F);
val it = true : bool
- print (OR F F);
val it = false : bool
```

```
val OR = fn b0 => fn b1 => b0 T b1
```

2. Considere a seguinte gramática para expressões booleanas:

$$e ::= b \texttt{ and } e \mid b \texttt{ or } e \mid b$$
$$b ::= 0 \mid 1$$

Responda aos itens abaixo, justificando sempre que necessário:

(a) (2 pontos) A gramática acima é ambígua? Se sim, apresente um exemplo de sentença com duas derivações distintas usando o espaço à direita.

A gramática não é ambígua

(b) (2 pontos) Qual é a associatividade das operações `and` e `or` nessa gramática?

Associativos à direita

(c) (2 pontos) Modifique a gramática de modo que a operação `and` tenha precedência maior que `or`.

```
e ::= b or e | f
f ::= b and f | b
b ::= 0 | 1
```

(d) (2 pontos) Escreva a **gramática original** em Prolog, utilizando a sintaxe de gramáticas lógicas.

(e) (2 pontos) Estenda a gramática em Prolog com atributos (argumentos) para calcular o valor lógico das expressões reconhecidas. Para este item, desconsidere precedência e associatividade: assuma que as operações são comutativas e possuem a mesma precedência.

Resposta da Questão 2d:

```
e --> b, [and], e.
e --> b, [or], e.
e --> b.

b --> [0].
b --> [1].
```

Resposta da Questão 2e:

```
e(0) --> b(0), [and], e(_).
e(B) --> b(1), [and], e(B).
e(1) --> b(1), [or], e(_).
e(B) --> b(0), [or], e(B).
e(B) --> b(B).

b(0) --> [0].
b(1) --> [1].
```

3. Nesta questão você deverá implementar, em SML/NJ, a função `len` que calcula o número de elementos presentes em uma lista usando diferentes técnicas de programação.

   (a) (2 Pontos) Implemente, em uma linha somente, a função `len0` sem usar casamento de padrões e sem usar `foldr` ou `foldl`. Você pode usar a função `tl: 'a list -> 'a list`.
   Importante: O tipo de sua função deve ser `''a list -> int`:

```
fun len0 L = if L = [] then 0 else 1 + len0 (tl L)
```

   (b) (2 Pontos) Implemente a função `len1` usando casamento de padrões, em duas linhas. O tipo de sua implementação deve ser `'a list -> int`:

```
fun len1 nil = 0
  | len1 (_::t) = 1 + len1 t
```

   (c) (2 Pontos) Implemente a função `len2:'a list -> int` usando a função `foldr`. Lembre-se: o tipo de `foldr` é `('a * 'b -> 'b) -> 'b -> 'a list -> 'b`. Sua implementação deve ter uma linha e não pode usar `map`.

```
fun len2 L = foldr (fn(_, b)=>b+1) 0 L
```

   (d) (2 Pontos) Implemente a função `len3:'a list -> int` usando uma combinação de `foldr` e `map:('a -> 'b) -> 'a list -> 'b list`. A operação de redução deve ser `(op +)`. Sua implementação deve ter uma linha.

```
fun len2 L = foldr (op +) 0 (map (fn _ => 1) L)
```

   (e) (2 Pontos) O tipo de `len0` deve ser `''a list -> int`, e o tipo de `len1` é `'a list -> int`. Qual a diferença entre esses tipos polimórficos?

```
'a é qualquer tipo; ''a não envolve números reais.
```