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 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 Extra (0.5 Pontos): formigas macho em geral não possuem asas. Cite mais duas famílias de insetos com elementos que não têm asas em qualquer momento da vida. Não vale citar dois elementos da mesma família como formiga saúva e formiga doceira.

1. A figura abaixo mostra uma parte do Triângulo de Pascal. O triângulo está organizado em uma matrix diagonal. A notação $(i, j), i \geq j$ denota a linha i e a coluna j da matriz:

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

   (a) (7 Pontos) Escreva, em ML, uma função `ptriangle`, de tipo `int` $*$ `int` $->$ `int`, que receba dois inteiros, i e j, e produza o elemento na posição (i, j) do Triângulo de Pascal. Por exemplo:

```
- ptriangle(2, 2);
val it = 1 : int
- ptriangle(5, 2);
val it = 10 : int
- ptriangle(0, 0);
val it = 1 : int
```

   (b) (3 Pontos) Qual é a complexidade assimptótica da função que você escreveu acima? Descreva essa complexidade em termos de i e j, os parâmetros passados para `ptriangle`.

2. Normalmente os valores alocados no registro de ativação de uma função não podem ser utilizados depois que essa função retorna. Linguagens que permitem esse tipo de uso dão margem a *bugs* de programação difíceis de serem encontrados.

   (a) (5 Pontos) Escreva um programa em C que contém um erro de programação. Tal erro deve ser causado pelo uso de uma posição de memória alocada durante a ativação de uma função, e não mais válida após o fim de tal ativação.

   (b) (5 Pontos) Ao contrário de C, ML permite que valores locais a uma função sejam usados após o retorno daquela função. Escreva um programa que evidencie essa propriedade da linguagem.

3. Reduções Beta são o mecanismo de computação usado no cálculo lambda. Uma Redução Beta enuncia que a aplicação $(\lambda x.e)y$ pode ser ré-escrita como $e[x \rightarrow y]$. Essa última notação significa que toda ocorrência de x deve ser substituída por y dentro da expressão e. Caso uma expressão lambda não possa mais sofrer reduções beta, diz-se que tal expressão está em *forma normal*.

   (a) (3 Pontos) Escreva a forma normal da expressão: $(\lambda x.\lambda y.xy)y$

   (b) (4 Pontos) Escreva uma expressão lambda que possua uma forma normal maior que seu tamanho original. Em outras palavras, uma vez reduzida, o tamanho dessa expressão lambda deve crescer. Mostre a expressão lambda original, e sua expressão em forma normal.

   (c) (3 Pontos) Escreva uma expressão lambda que não tenha forma normal.

4. *Idris* é uma linguagem de programação funcional. Sintaticamente, Idris é parecida com *Haskell*, outra linguagem funcional. Pragmaticamente, ambas essas linguagens se parecem com ML, a linguagem funcional vista no curso DCC024. Idris possui um tipo `List`, que pode ser usado da seguinte forma:

```
Idris> [1,2,3]
[1, 2, 3] : List Integer
Idris> ["a", "b", "c"]
["a", "b", "c"] : List String
Idris> [1,2,3] ++ [4,5,6]
[1, 2, 3, 4, 5, 6] : List Integer
Idris> [2.76] ++ [3.14, 6.28]
[2.76, 3.14, 6.28] : List Double
Idris> 2.76 :: [3.14, 6.28]
[2.76, 3.14, 6.28] : List Double
```

O operador `++` é declarado da seguinte forma:

```
(++) : List a -> List a -> List a
(++) Nil       ys = ys
(++) (x :: xs) ys = x :: xs ++ ys
```

(a) (3 Pontos) A implementação de `List` utiliza um tipo de polimorfismo. Que tipo de polimorfismo é este?

(b) (3 Pontos) Escreva o tipo do operador `::`

(c) (4 Pontos) A gramática abaixo descreve expressões sobre listas:

$$\begin{array}{rcl} \langle L \rangle & ::= & \langle L \rangle :: \langle L \rangle \\ & | & \langle L \rangle ++ \langle L \rangle \\ & | & [\texttt{x}] \\ & | & \texttt{x} \end{array}$$

Essa gramática permite a criação de expressões como `[x]::[x]`, `[x]::x` ou `x++x`,, que não são válidas, pois vão contra os tipos de `::` e `++`. Modifique a gramática acima para que ela permita somente a criação de expressões válidas.

5. Esta questão refere-se à implementação do tipo algébrico árvore. Essa implementação pode ser vista logo abaixo:

```
datatype Tree = Leaf | Node of Tree * int * Tree
```

(a) (3 Pontos) Qual é o tipo da função `toList`, cuja implementação pode ser vista logo abaixo?

```
fun toList Leaf = nil
  | toList (Node (L, e, R)) = e :: toList L @ (toList R)
```

(b) (7 Pontos) Implemente uma função `rm`, de tipo `int -> int Tree -> int Tree`, que remove todas as ocorrências de um inteiro `e` de uma instância de `'a tree`. Você pode usar as seguintes funções em sua implementação:

```
val ins = fn : int -> int Tree -> int Tree
val union = fn : int Tree -> int Tree -> int Tree
```

A função `ins` recebe um elemento `e` e uma árvore `t`, e produz uma nova árvore contendo `e` e todos os elementos de `t`. A função `union`, por sua vez, recebe duas árvores, `t0` e `t1`, e produz uma nova árvore que contém todos os elementos de `t0` e de `t1`.

Abaixo, vêem-se alguns exemplos de uso:

```
- val t1 = ins 2 Leaf ;
val t1 = Node (Leaf,2,Leaf) : Tree
- val t2 = ins 4 Leaf ;
val t2 = Node (Leaf,4,Leaf) : Tree
- toList t2 ;
val it = [4] : int list
- val t3 = union t1 t2;
val t3 = Node (Node (Leaf,2,Leaf),4,Leaf) : Tree
- val t4 = rm 2 t3 ;
val t4 = Node (Leaf,4,Leaf) : Tree
```