Primeira Prova de Linguagens de Programação
- DCC024 -
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. São perguntas: "Posso fazer uma pergunta?"ou "Quanto tempo falta?"
- 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	Extra

Questão Extra (0.5 Pontos): "*Na Cacuânia se deve ser cacuano*". Onde ficava a Cacuânia? Pede-se o nome atual do país onde ficava a Cacuânia, ou o nome atual de um país fronteiriço dessa região.

1. Existem vários *paradigmas de programação*. Diferentes autores inclusive defendem listas diferentes. Contudo, a maior parte dos teóricos da área de linguagem de programação entende que existe um paradigma imperativo e um paradigma funcional.

   (a) (1 Ponto) Qual o modelo computacional que fundamenta o paradigma imperativo? Um modelo computacional é uma abstração que determina quais algoritmos podem ser escritos naquele paradigma, como esses algoritmos poderiam ser escritos e qual a complexidade assintótica desses algoritmos. A título de exemplo, diferentes modelos computacionais já foram usados para mostrar que o "*Problema da Parada*"não possui solução computacional.

   (b) (1 Ponto) Qual o modelo computacional que fundamenta o paradigma de programação funcional?

   (c) (2 Pontos) Por que o paradigma imperativo possui esse nome?

   (d) (2 Pontos) Cite uma característica importante do paradigma funcional que o distingue do paradigma imperativo.

   (e) (4 Pontos) O paradigma de programação é uma característica da linguagem de programação ou dos programas que são escritos naquela linguagem? Defenda seu ponto de vista.

2. Uma linguagem de programação é dita *fortemente tipada* caso programas escritos nessa linguagem não fiquem em estado indefinido. Note que esse conceito, de tipagem fraca ou forte, não é binário: uma linguagem pode ser mais ou menos fortemente tipada que outra: a linguagem mais fortemente tipada terá menos situações que geram comportamento indefinido.

   (a) (1 Pontos) Uma forma de garantir a ausência de comportamento indefinido é via *verificações em tempo de execução*. Essas verificações são testes condicionais *implícitos*. O programa abaixo mostra um teste desse tipo, que é inserido em SML/NJ:

O que o programador escreveu:

```
fun first L = hd L
```

O que o compilador produziu:

```
fun first L =
  if null L
  then raise Empty
  else hd L
```

Exemplos de execução:

```
- first ["oi"];
val it = "oi" : string
```

   O código mais a direita é um exemplo de programa que usa a função `first`. Escreva um programa diferente que usa `first` que poderia chegar a um estado indefinido caso a verificação implícita no código produzido pelo compilador não existisse.

   (b) (3 Pontos) O compilador java (`javac`) vai inserir um teste implícito no programa abaixo, a fim de evitar que o programa chegue a um estado indefinido. O que verifica esse teste?

O que o programador escreveu:

```
void testImplicitIsNull(Element e) {
  System.out.println(e.toString());
}
```

   (c) O compilador java (`javac`) vai inserir três testes implícitos no programa abaixo, a fim de evitar que o programa chegue a um estado indefinido. O que verifica cada um desses testes?

O que o programador escreveu:

```
int boundsCheck(int[] array, int index) {
  return array[index];
}
```

   i. (2 Pontos) Teste 1:

   ii. (2 Pontos) Teste 2:

   iii. (2 Pontos) Teste 3:

3. O objetivo desta questão é implementar uma busca em lista que retorna a primeira posição da lista em que certa condição ocorra. Para tanto, iremos implementar a função em quatro partes.

   (a) (3 Pontos) Escreva uma função `index`, de tipo `int -> 'a list -> (int * 'a) list`, tal que `index n L` cria uma lista `L'` de tuplas indexadas a partir de `n`. Por exemplo:

   ```
   - index 2 ["a", "b", "c"];
   val it = [(2,"a"),(3,"b"),(4,"c")] : (int * string) list
   - index 0 ["a", "b", "c"];
   val it = [(0,"a"),(1,"b"),(2,"c")] : (int * string) list
   ```

   Note que cada elemento de `L'` é um par (i, e), sendo i o índice do elemento e de `L`.

   (b) (2 Pontos) Escreva uma função `count`, de tipo `'a list -> (int * 'a) list`, que transforme uma lista `L` em uma lista de tuplas indexadas a partir de zero (fique à vontade para usar a função `index` da questão anterior. Caso não a tenha feito, assuma sua existência). Exemplos:

   ```
   - count [2, 3, 5, 7];
   val it = [(0,2),(1,3),(2,5),(3,7)] : (int * int) list
   - count [true, false];
   val it = [(0,true),(1,false)] : (int * bool) list
   - count ["a", "b", "c"];
   val it = [(0,"a"),(1,"b"),(2,"c")] : (int * string) list
   ```

   (c) (3 Pontos) Escreva a função `find_aux`, de tipo `('a -> bool) -> (int * 'a) list -> int`, que receba um predicado (uma função que retorna um boleano) e uma lista indexada (uma lista de pares (i, e), em que i é o índice do elemento e), e retorne o índice i do primeiro elemento e que torna verdadeiro o predicado. Exemplos:

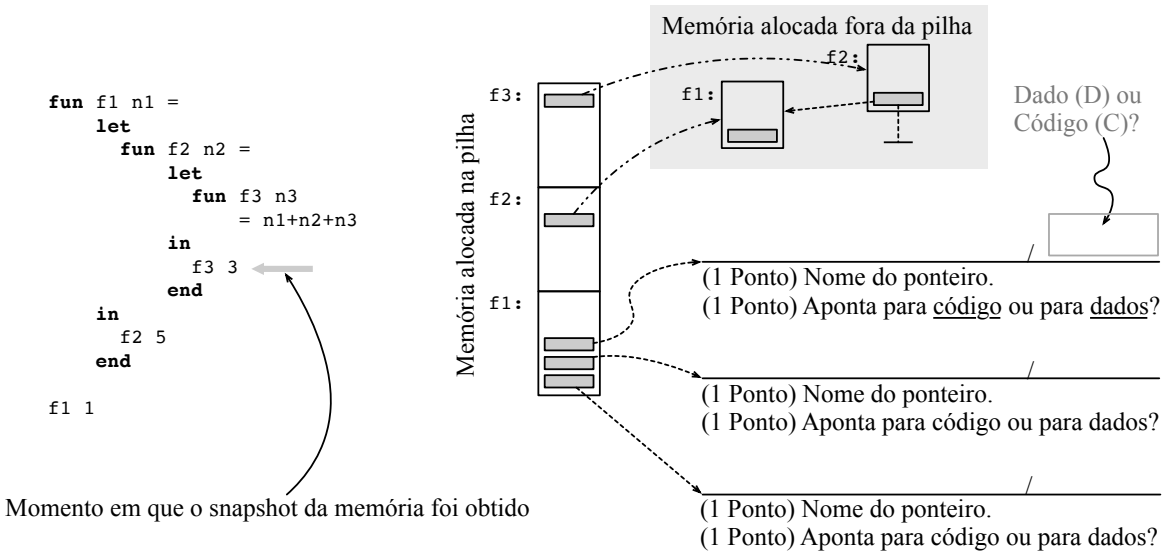
   ```
   - find_aux (fn x => x mod 2 <> 0) [(0,2),(1,3),(2,5)];
   val it = 1 : int
   - find_aux (fn x => x > 4) [(0,2),(1,3),(2,5)];
   val it = 2 : int
   - find_aux (fn x => x > 6) [(0,2),(1,3),(2,5)];
   val it = ~1 : int
   ```

   (d) (2 Ponto) Escreva uma função `find`, de tipo `('a -> bool) -> 'a list -> int`, tal que `find f L` retorne o índice do primeiro elemento de `L` que seja verdadeiro para o predicado `f`. Exemplos:

   ```
   - find (fn x => size(x) > 2) ["Eu", "amo", "voce"];
   val it = 1 : int
   find (fn x => x mod 2 = 0) [1, 3, 4, 5];
   val it = 2 : int
   ```

   Dica: use `find_aux`. Você pode assumir que a função existe, caso não a tenha feito anteriormente.

4. Essa questão diz respeito ao programa escrito em SML/NJ que aparece na parte esquerda da figura abaixo.



(a) (6 Pontos) A figura mostra o registro de ativação das diferentes funções chamadas para calcular `f1 1`. A foto da memória (*snapshot*) foi tirada quando a chamada `f3 3` ocorreu. Os registros de ativação das diferentes funções contém a mesma estrutura: espaços para alocar os dados que cada função precisa para executar corretamente. Dentre esses dados, SML/NJ aloca ao menos três ponteiros em cada registro de ativação. Pede-se que você escreva o nome desses três ponteiros, e indique, para cada ponteiro, se ele aponta para uma área de código (para algum endereço de instrução de máquina) ou para uma área de dados (uma região que contém os dados que o programa precisa para funcionar: pilha, *heap*, etc). Caso você não saiba o nome do ponteiro, não se preocupe, você pode usar o espaço abaixo para explicar para quê aquele ponteiro serve.

(b) Os registros de ativação são alocados em um espaço de dados que funciona como uma pilha: o último registro alocado será o primeiro a ser desalocado, tão logo a função que o criou retorne. Contudo, em SML/NJ alguns dados de registro de ativação não podem ser alocados na pilha: eles podem ser necessários após a função que os alocou retornar.

i. (2 Pontos) Quais dados de uma função não podem ser alocados na pilha?

ii. (2 Pontos) Escreva um programa (simples) que demonstra que dados podem ser necessários após a função que os criou retornar.