Prova Final 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.
- O vasto esforço despendido para tornar essa prova difícil mostrou-se infrutífero, então aproveite, pois ela está divertida!

Tabela 1: Pontos acumulados (para uso do instrutor)

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

1. Essa questão refere-se à alocação de arranjos nas linguagens C e Java.

   (a) (3 pontos) A linguagem C, bem como várias outras linguagens de programação imperativas, possui arranjos cujo tempo de acesso é aleatório. O acesso direto a um elemento `a[i][j]` em um arranjo bidimensional em C é possível porque o compilador transforma tal acesso em um único acesso à memória. Qual é a fórmula que o compilador usa para fazer uma referência direta ao elemento `a[i][j]`, sabendo que o arranjo foi declarado como `int a[M][N]`? Escreva essa fórmula em função das variáves `M`, `N`, `i` e `j`. Algumas dessas variáveis podem não fazer parte da expressão final.

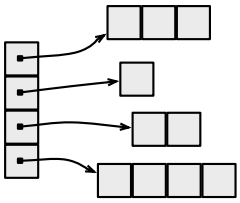
   (b) (4 pontos) A função `printRowMajor`, à direita, é um pouco mais eficiente que a função `printColumnMajor`, à esquerda. É possível que implementássemos um compilador de C em que a função `printColumnMajor` fosse mais eficiente. O que esse compilador teria de fazer?

```
void printRowMajor() {
  int[M][N] a;
  int i, j;
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      printf("%d ", a[i][j]);
    }
  }
}
```

```
int printColumnMajor() {
  int[M][N] a;
  int i, j;
  for (int j = 0; j < 3; j++) {
    for (int i = 0; i < 3; i++) {
      printf("%d ", a[i][j]);
    }
  }
}
```

   (c) (3 pontos) Arranjos em Java não são alocados exatamente como em C. Elementos em uma linha (em uma mesma dimensão) são alocados contiguamente, mas linhas separadas não ficam em posições contíguas de memória:

```
void foo() {
  int[][] a;
  a = new int[4][];
  a[0] = new int[3];
  a[1] = new int[1];
  a[2] = new int[2];
  a[3] = new int[4];
}
```



   Dada essa forma de alocação de dados, quantos acessos à memória são necessários para que possamos ler o elemento `a[i][j]` em um arranjo de Java? Justifique a sua resposta.

2. Essa questão refere-se aos diferentes espaços de alocação de dados que são criados pelo compilador, para garantir que as variáveis de um programa possam ser devidamente armazenadas durante a execução do programa.

   (a) (2 pontos) Escreva um programa em C que possua algum dado alocado em memória estática. Explique qual dado é esse.

   (b) (3 pontos) Escreva um programa em Python que possua algum dado alocado no *heap*. Explique qual dado é esse.

   (c) (3 pontos) Escreva um programa em SML que possua algum dado alocado na pilha. Explique qual dado é esse.

   (d) (2 pontos) Historicamente, qual desses mecanismos de alocação de dados – memória estática, heap e pilha – surgiu primeiro? Justifique a sua resposta informando uma linguagem que já possuia o mecanismo escolhido, antes das duas outras formas se tornarem também populares.

3. Essa questão refere-se ao bloco de código abaixo, escrito na linguagem de programação Python. Você deverá escrever suas respostas, nas questões (a) e (b), dentro da área delimitada. Você não pode escrever código que veja a ser executado antes ou depois dessas áreas:

```
class Rectangle:
  def __init__(self, cor, x, y, largura, altura):
    self.cor = cor
    self.x = x
    self.y = y
    self.largura = largura
    self.altura = altura
```

```
def contaCores(recLista, cor):
```

(a) (4 pontos) Implemente o corpo do método `contaCores`, que recebe uma lista de retângulos, e retorna um inteiro informando quantos retângulos possuem a cor escolhida.

```
recLista = [
             Rectangle("blue", 10, 20, 10, 20),
             Rectangle("yellow", 10, 20, 2, 5),
             Rectangle("blue", 10, 20, 10, 20)
           ]
print contaCores(recLista, "blue")
```

```
```

(b) (3 pontos) Escreva código neste espaço, de modo que a implementação do método `hasLargeArea` funcione corretamente. Para isso, a classe `Rectangle` deve respeitar o contrato imposto por `hasLargeArea`.

```
def hasLargeArea(rect, limiar):
  if rect.getArea() > limiar:
    return True
  else:
    return False;
```

```
def secret(listRect, limiar):
  return [rec for rec in listRect if hasLargeArea(rec, limiar)]
```

(c) (3 pontos) Explique o que faz o método `secret`. Comece descrevendo quais são os tipos das variáveis de entrada, `listRect` e `limiar` e o tipo do valor retornado pelo método.

4. (10 Pontos) Considere o seguinte jogo: tem-se um tabuleiro de $N \times N$ casas, e deseja-se dispor N cores nesse tabuleiro N vezes, de modo que cada linha e cada coluna possuam somente cores diferentes. O predicado abaixo, escrito em Prolog, resolve esse problema para um tabuleiro 2×2, considerando-se as cores verde e azul:

```
legal([]).
legal([(X, Y, C) | Rest]) :-
  legal(Rest),
  member(X, [1, 2]),
  member(Y, [1, 2]),
  member(C, [azul, verde]),
  nocheck((X, Y, C), Rest).
```

Abaixo vê-se uma possível execução do predicado `legal`:

```
?- L = [(1, _, verde), (2, _, verde), (1, _, azul), (2, _, azul)], legal(L).
L = [ (1, 1, verde), (2, 2, verde), (1, 2, azul), (2, 1, azul)] ;
L = [ (1, 2, verde), (2, 1, verde), (1, 1, azul), (2, 2, azul)] ;
false.
```

O predicado `legal` utiliza um predicado auxiliar, `nocheck`. Temos que `nocheck((X, Y, C), Rest)` é verdade quando a cor `C`, colocada no quadrado `(X, Y)`, não conflita com qualquer outra cor colocada na linha `X` ou na coluna `Y` que por ventura estejam representadas na lista `Rest`. Nesta questão, implemente esse predicado, tendo em vista as seguintes observações:

- Duas cores iguais precisam estar em linhas e colunas diferentes.
- Duas cores diferentes não podem estar na mesma linha e coluna.

5. Uma linguagem de programação é dita *estrita* se os parâmetros passados para as funções sempre são avaliados antes dessas funções serem chamadas. Por exemplo, SML é uma linguagem estrita. Isso quer dizer que as expressões `1 + 2` e `x * 4`, no programa abaixo, serão avaliadas antes da função `foo` ser chamada.

```
- fun foo(x, y) = if x > y then x else y;
- val x = ~1;
- foo(1 + 2, x * 4);
val it = 3 : int
```

(a) (1 Pontos) A passagem de parâmetros por expansão de macros é um mecanismos estrito ou não?

(b) (4 Pontos) Justifique a sua resposta para o item (a) acima, mediante um exemplo, escrito na linguagem de programação C. Explique porque o seu exemplo é uma justificativa válida.

(c) (4 Pontos) Mostre, via um programa escrito em Python, que essa linguagem de programação é estrita. Explique o comportamento esperado para esse programa.

(d) (1 Pontos) Existe alguma linguagem de programação cuja avaliação de parâmetros passados para funções é exclusivamente não estrita?

6. Essa questão busca relembrar conceitos relacionados à linguagem de programação SML.

   (a) (2 pontos) Aponte uma semelhança entre as linguagens de programação Python e SML.

   (b) (2 pontos) Aponte uma semelhança entre as linguagens de programação SML e Python. A resposta desse item deve ser diferente da resposta dada no item anterior.

   (c) (6 pontos) Escreva uma função `cycle`, em SML, de tipo `'a list * int -> 'a list` que receba uma lista e um número inteiro e retorne a mesma lista, mas com os n primeiros elementos movidos para o final da lista. Por exemplo,`cycle ([1, 2, 3, 4, 5, 6], 2)` deve retornar `[3, 4, 5, 6, 1, 2]`.