Prova Final 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 6

1. Essa questão refere-se a algumas decisões de projeto que foram tomadas por ocasião da definição da sintaxe de operadores em SML.

   (a) (5 Pontos) Considere a sequência de chamadas logo abaixo:

```
- foldr (op +) 0 [2,3,4,5,6];
val it = 20 : int

- map not [true, false, true];
val it = [false,true,false] : bool list

- map ~ [2,3,4,5];
val it = [~2,~3,~4,~5] : int list

- map (op * ) [(3.1, 2.7), (~3.14, 9.12)];
val it = [8.37,~28.6368] : real list
```

   Por que é necessário prefixar os operadores de adição e multiplicação com o transformador `op`, e não é necessário fazer isso com as negações lógica e aritmética?

   (b) (5 Pontos) SML é uma linguagem com poucas palavras reservadas. Podemos, por exemplo, sobre-escrever o operador de adição +:

```
- infix 3 +;
infix 3 +

- fun op + (a, b) = a - b;
val + = fn : int * int -> int

- 3 + 2;
val it = 1 : int
```

   Curiosamente, as palavras `andalso` e `orelse` são reservadas na linguagem. Veja as consequências dessa decisão:

```
- foldr (fn(x, y) => x orelse y) false [false, true, false];
val it = true : bool

- foldr (op orelse) false [false, true, false];;
stdIn:9.11-9.18 Error: syntax error: deleting  ORELSE RPAREN
```

   Por que foi necessário adortar-se essa decisão que diminui a ortogonalidade da linguagem?

2. Os métodos logo abaixo estão implementados em Java. Esta questão refere-se a essas duas implementações:

```
public static void alimentaAnimais(Set<Animal> set) {
  for (Animal a : set) {
    a.eat();
  }
}

public static void alimentaCachorros(Set<Cachorro> set) {
  for (Cachorro d : set) {
    d.eat();
  }
}
```

(a) (4 Pontos) Se assumirmos que a classe `Cachorro` estende a classe `Animal`, então qual das implementações acima é melhor? Uma implementação é melhor que a outra quando ela adere de forma mais clara a bons princípios de programação. Neste caso, pede-se que sejam comparadas as implementações de `alimentaAnimais` e de `alimentaCachorros`. Você deve justificar a sua resposta.

(b) (6 Pontos) Comparemos agora, as mesmas duas funções, desta vez implementadas em Python:

```
class Animal:
  def __init__(self, name):
    self.name = name
  def eat(self):
    print self.name + " is easting"

class Cachorro(Animal):
  def eat(self):
    print self.name + " is eating like a dog"
```

```
def alimentaAnimais(setAnimal):
  for a in setAnimal:
    a.eat()

def alimentaCachorros(setCachorro):
  for c in setCachorro:
    c.eat()
```

Do ponto de vista de boas práticas de programação, qual destas funções é a melhor delas, `alimentaAnimais` ou `alimentaCachorros`? Lembre-se de justificar a sua resposta.

3. O princípio da substituição de Liskov reza que "em situações em que se espera um tipo T, pode-se passar um tipo S, se S for subtipo de T". Esse é um dos princípio mais básicos a nortear a programação orientada a objetos. Em outras palavras, qualquer linguagem orientada a objetos atende, de alguma forma, o princípio da substituição de Liskov. Curiosamente, algumas construções que usam polimorfismo paramétrico são proibidas em Java, ainda que elas aparentemente atendam ao princípio da substituição. Por exemplo, a atribuição abaixo, na função `main`, não é permitida pelo compilador Java, ainda que a classe `Dog` estenda a classe `Animal`:

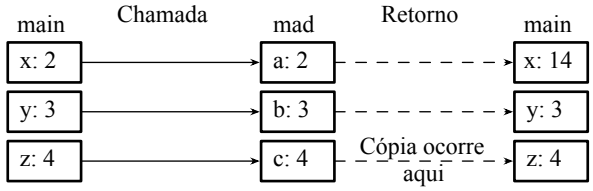
```
class Animal() {...}

class Dog() extends Animal {...}

main() {
  List<Animal> l = new List<Dog>();
}
```

   (a) (8 Pontos) Por que tal atribuição não é permitida pela linguagem Java? Ilustre a sua resposta com um exemplo de código que não funcionaria corretamente caso a atribuição fosse permitida pela linguagem.

   (b) (2 Pontos) Esse tipo de situação não é um problema na linguagem Python. Porque programadores Python não precisam se preocupar com esse tipo de atribuição entre estruturas de dados parametrizadas por tipo?

4. (10 Pontos) Existe um mecanismo de passagem de parâmetro chamado *valor-resultado*. Neste sistema, o valor do parâmetro real é copiado para o parâmetro formal no momento em que uma função é chamada, tal como se dá na chamada por *valor*. Finda a execução da função invocada, o valor calculado no parâmetro formal é copiado de volta para o parâmetro real. A figura abaixo ilustra essa técnica de passagem de parâmetros. Neste exemplo, serão impressos os valores `14, 3` pelo programa usado como exemplo, assumindo-se que o parâmetro `a` da função `mad` é passado por valor resultado.

```
void mad(val-res int a, int b, int c) {
    b = b + c;
    a = a * b;
}
void main() {
    int x = 2;
    int y = 3;
    int z = 4;
    mad(x, y, z);
    print("%d, %d\n", x, y);
}
```



A passagem por valor-resultado é bastante parecida com a passagem por referência. Existem, contudo, diferenças. Nesta questão, você deve escrever um programa, em pseudo-código, que ilustre a diferença entre a passagem por resultado, e a passagem por referência. Você precisa explicar claramente qual seria o resultado do programa, se alguns parâmetros fosse passados por referência ou por valor-resultado.

5. Nesta questão você deverá implemente, em Prolog, o predicado `isWord(L, N, A)`, que é verdade sempre que `L` for uma lista de tamanho `N`, formada por elementos do alfabeto `A`. Por exemplo:

```
?- isWord(L, 3, [0, 1]).
L = [0, 0, 0] ;
L = [0, 0, 1] ;
L = [0, 1, 0] ;
L = [0, 1, 1] ;
L = [1, 0, 0] ;
L = [1, 0, 1] ;
L = [1, 1, 0] ;
L = [1, 1, 1] ;
false.

?- isWord(L, 2, [a, b, c]).
L = [a, a] ;
L = [a, b] ;
L = [a, c] ;
L = [b, a] ;
L = [b, b] ;
L = [b, c] ;
L = [c, a] ;
L = [c, b] ;
L = [c, c] ;
false.
```

   (a) (8 Pontos) Implemente o predicado `isWord(L, N, A)`:

   (b) (2 Pontos) Use o predicado `findAll`, combinador com `isWord`, para listar todos os números binários que possuam até três dígitos. Não se preocupe em eliminar zeros a esquerda. Não se preocupe em perguntar a sintaxe correta de `findAll` para o instrutor, pois ele não responderá.

6. Essa questão refere-se à classe `Staff`, cuja implementação, em Python, é mostrada logo abaixo:

```
class Staff:
  payroll = {}
  def getSalary(self, name):
    if self.payroll.has_key(name):
      return self.payroll[name]
    else:
      return 0.0
  def addEmp(self, name, salary):
    self.payroll[name] = salary
  def raiseSalary(self, name, salary):
    self.payroll[name] = self.payroll[name] + salary
```

(a) (3 Pontos) O método `getSalary` utiliza um valor especial, 0.0 como o salário de um empregado inexistente. Qual a desvantagem desta forma de tratamento de erros?

(b) (3 Pontos) Crie uma classe de exceções `NonExistentEmployee`, em Python, para tratar a situação excepcional de uma busca sobre um empregado inexistente.

(c) (4 Pontos) Modifique o método `getSalary` para disparar uma exceção do tipo `NonExistentEmployee` caso o nome solicitado não possua uma entrada no banco de dados.