Prova Final de Linguagens de Programação
- DCC024B -
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.

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.
- Serão avaliadas as seis melhores respostas. Então sinta-se livre para abandonar alguma questão devido ao tempo.
- A prova não é difícil, ela é divertida, então aproveite!

Tabela 1: Pontos acumulados

Questão 1	Questão 2	Questão 3	Questão 4	Questão 5	Questão 6	Questão 7

1. As linguagens de programação podem ser divididas em dois grupos principais: as linguagens *imperativas* e as linguagens *declarativas*.

   (a) (4 pontos) Algumas linguagens favorecem a programação declarativa, enquanto outras favorecem a programação imperativa. Por outro lado, pode-se programar declarativamente ou imperativamente na maioria das linguagens de programação. Escolha um certo algoritmo, com o qual você esteja familiar, e o escreva em pseudo-código, de forma imperativa e declarativa.

   (b) (3 pontos) Descreva uma vantagem da abordagem declarativa sobre a abordagem imperativa. Não é preciso ater-se ao exemplo da questão anterior.

   (c) (3 pontos) Agora faça o oposto: mostre uma vantagem da abordagem imperativa sobre a sua contra-parte declarativa.

2. Esta questão refere-se à passagem de parâmetros por nome. Este método causa a avaliação do parâmetro real sempre, e somente se, ele for utilizado no corpo da função chamada.

   (a) (5 pontos) Para evitar a captura de variáveis, um fenômeno que ocorre na passagem de parâmetros por expansão de macros, a avaliação do parâmetro formal, na passagem por nomes, utiliza o contexto do chamador. Escreva um programa, em pseudo-código, que retornaria resultados diferentes, dependendo do método de chamada ser expansão de macros ou passagem por nome.

   (b) (5 pontos) A passagem por nomes difere da avaliação preguiçosa por que esta última utiliza um cache para evitar a múltipla avaliação de parâmetros. Escreva um programa, em pseudo-código, que mostre de forma irrefutável a diferença entre estes dois mecanismos de passagem de parâmetros.

3. (10 pontos) Considere o seguinte programa, implementado em SML:

```
exception NegArgumentException of int;
exception PrecisionException of int * int;
val PRECISION = 1073741823
fun fact 0 = 1
  | fact n = n * fact (n - 1)
fun factEx n = if n < 0
               then raise NegArgumentException n
               else if n > 12
                    then raise PrecisionException (n, PRECISION)
                    else fact n
fun useFact n =
  "Answer = " ^ Int.toString (factEx n)
  handle NegArgumentException n =>
    "Argument is negative: " ^ Int.toString n
  handle PrecisionException (n, maxInt) =>
    "Fact of " ^ Int.toString n ^ " bigger than " ^ Int.toString maxInt
```

Implemente este programa em Java. A sua implementação deve ser fiel à semântica do programa original tanto quanto possível, em particular, ela deve tratar os dois possíveis tipos de exceções.

4. (2 pontos por questão) Para cada predicado abaixo, informe sua complexidade assimptótica, em termos de tempo de execução e espaço ocupado na pilha de registros de ativação. A complexidade deve ser informada de forma não ambígua; por exemplo: "a função é cúbica no número de elementos de seu primeiro parâmetro".

   - `hd([H|_], H).`

   - ```
     append([],X,X).
     append([Head|Tail], X, [Head|Suffix]) :-  append(Tail, X, Suffix).
     ```

   - ```
     reverse([],[]).
     reverse([Head|Tail],Rev)
      :-  reverse(Tail,TailRev),  append(TailRev,[Head],Rev).
     ```

   - ```
     reverse(X,Y) :- rev(X,[],Y).
     rev([],Sofar,Sofar).
     rev([Head|Tail],Sofar,Rev)
      :-  rev(Tail,[Head|Sofar],Rev).
     ```

   - ```
     xequals([], []).
     xequals([H1|T1], [H2|T2]) :- H1 = H2, xequals(T1, T2).
     ```

5. Considere o programa Java abaixo. Nas cinco próximas perguntas, as respostas possíveis são:

   - Será impresso "...".
   - O programa não será compilado.
   - Ocorrerá uma falha em tempo de execução.

```
1   class Animal { void eat() { System.out.println("Animal eats."); } }
2   class Fish extends Animal {
3     void eat() { System.out.println("Fish eats."); }
4     void swim() { System.out.println("Fish swims."); }
5   }
6   class Shark extends Fish { void eat() { System.out.println("Shark eats."); } }
7   public class Dispatch {
8     public static void main(String args[]) {
9       Animal a = new Fish();
10      Fish f = new Shark();
11      a.eat();
12      a.swim();
13      f.eat();
14      f.swim();
15      Shark s = (Shark)f;
16      s.eat();
15    }
16  }
```

   (a) (1 ponto) O que acontecerá na linha `11`?

   (b) (1 ponto) O que acontecerá na linha `12`?

   (c) (1 ponto) O que acontecerá na linha `13`?

   (d) (1 ponto) O que acontecerá na linha `14`?

   (e) (1 ponto) O que acontecerá na linha `16`?

   (f) (5 pontos) Desenhe as tabelas virtuais dos objetos `a`, `f` e `s` imediatamente depois da execução da linha `14` do programa. Mostre, com setas, quais métodos são apontados por cada entrada da tabela.

6. (10 pontos) Duas *strings* são *anagramas* quando elas são formadas pelos mesmos caracteres, não necessariamente na mesma ordem. Implemente o predicado `solve(S, A)`, que é verdade quando os átomos `S` e `A` são anagramas. Por exemplo:

```
?- solve(ana, naa).
true.

?- solve(ana, nab).
false.

?- solve(ana, nnaa).
false.
```

Neste exercício talvez você queira usar o predicado `name(A, L)`, que é verdade quando `L` é a lista de caracteres ASCII que forma o átomo `A`. Por exemplo:

```
?- name(abc, L).
L = [97, 98, 99].
```

7. (10 pontos) Costuma-se dizer que Java é uma linguagem orientada por objetos, porém, a orientação por objetos, enquanto uma filosofia de desenvolvimento de *software*, é muito mais uma característica do programa, que da linguagem de programação. Considere, por exemplo, o programa abaixo. Este programa, embora escrito em Java, não segue qualquer princípio de codificação orientado por objetos. Neste exercício você deve re-escrever o programa para que ele seja mais "orientado por objetos".

```
class Node { String data; Node link; }
class Stack { Node top; }
public class Controller {
  public static void add(Stack s, String data) {
    Node n = new Node();
    n.data = data;
    n.link = s.top;
    s.top = n;
  }
  public static boolean hasMore(Stack s)
    { return s.top != null; }
  public static String remove(Stack s) {
    Node n = s.top;
    s.top = n.link;
    return n.data;
  }
}
public class Main {
  public static void main(String args[]) {
    Stack s = new Stack();
    Controller.add(s, "AA");
    Controller.add(s, "BB");
    while (Controller.hasMore(s)) {
      String out = Controller.remove(s);
      System.out.println(out);
    }
  }
}
```