Exame Especial de Linguagens de Programação

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 cinquenta minutos após seu início. O instrutor avisará quando faltarem somente 15 minutos para o final do exame.
- 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.
- Serão avaliadas somente as seis melhores respostas. Então sinta-se livre para abandonar alguma questão devido ao tempo.
- H'a quem diga que estas provas são divertidas, então aproveite!

Tabela 1: Pontos acumulados (para uso do instrutor)

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

1. Em SML podemos representar conjuntos como listas que não possuem elementos repetidos.

   (a) (6 pontos) Implemente em SML uma função `union` de tipo `''a list list -> ''a list` que receba uma lista de conjuntos, isto é, uma lista de listas, e retorne a união de todos estes conjuntos. Por exemplo:

   ```
   - union [[1,2], [], [4,5], [4,5]];
   val it = [1,2,4,5] : int list

   - union [["a", "ab"], ["ab", "c"], ["def"]];
   val it = ["a","ab","c","def"] : string list

   - union [[(1,2), (2, 4)], [(1,2)]];
   val it = [(2,4),(1,2)] : (int * int) list
   ```

   A propósito: lembre-se de remover elementos repetidos.

   (b) (2 pontos) O tipo da função `union` é `''a list list -> ''a list`. O que quer dizer o duplo apostrofo no parâmetro do construtor de tipos?

   (c) (2 ponto) Qual é a complexidade assimptótica da função `union` implementada na letra (a) desta questão?

2. (10 pontos) Nesta questão você deverá implementar uma gramática lógica em Prolog que reconheça linguagens cujas *strings* têm a forma $a^n b^n c^n$. Frases válidas nesta gramática são, por exemplo, *abc*, *aabbcc* e *aaabbbccc*. Por outro lado, *abcc* não é uma frase válida nesta gramática, pois o número de caracteres *c* difere do número de *a*'s e *b*'s. Note que esta linguagem não pode ser reconhecida por uma gramática livre de contexto. Você precisará embutir atributos na gramática lógica. Para ajudá-lo a lembrar a sintaxe, abaixo é dada uma gramática lógica em Prolog que reconhece números inteiros simples:

```
number --> digit, number.
number --> digit.

digit --> [0] ; [1] ; [2] ; [3] ; [4] ; [5] ; [6] ; [7] ; [8] ; [9].
```

3. Seja `Time` uma classe, implementada em Python, que descreve um *tempo*. O tempo, neste caso, é um objeto com três campos: hora, minuto e segundo. Uma parte da classe `Time` é dado logo abaixo:

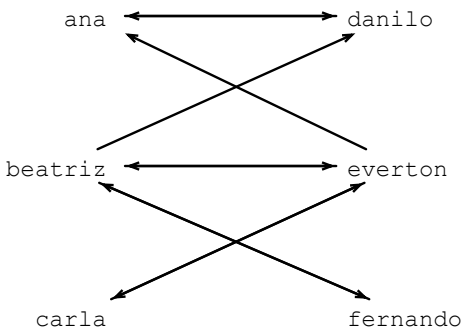
```
class BadTimeException(Exception):
  """This exception is used everytime the user enters an invalid number for
     minutes or seconds."""
  def __init__(self, message):
    self.value = message
  def __str__(self):
    return repr(self.value)

class Time:
  """This class encodes a time, which is defined by a number of hours, minutes
     and seconds."""
  def __init__(self, h, m, s):
    """IMPLEMENTAR"""
  def __str__(self):
    return strTm(self.hours) + ':' + strTm(self.mins) + ':' + strTm(self.secs)
  def add(self, t):
    """IMPLEMENTAR"""
```

(a) (5 pontos) Implemente o método `__init__` da classe `Time`. Este método deve disparar uma instância de `BadTimeException` caso a hora (`h`), o minuto (`m`) ou o segundo (`s`) estiverem em um formato impróprio. Horas são inteiros não negativos. Minutos e segundos são inteiros entre zero e 60.

(b) (5 pontos) Implemente o método `add` que receba uma outra instância de `Time`, chamada `t`, e some o tempo em `t` com o tempo do objeto corrente. Este método não deve retornar nenhum valor, mas deve ter o efeito colateral de modificar o tempo do objeto corrente. O método `add` deve preservar a invariante que os campos segundos e minutos são inteiros entre zero e sessenta inclusive.

4. (10 pontos) Dado um grafo, um casamento perfeito entre vértices é um pareamento dos vértices desse grafo de tal forma que todo vértice receba um, e somente um par. Aproveitando a época, vamos escrever um predicado `santantonio` em Prolog que tente realizar um casamento perfeito entre um grupo de pessoas. Vamos usar o predicado `gosta(X, Y)` para denotar a idéia de que `X` e `Y` são pessoas, e que `X` gosta de `Y`. Note que este predicado é unidirecional, pois `gosta(X, Y)` não implica em `gosta(Y, X)`. Por exemplo, na figura abaixo temos uma relação de afinidade entre um grupo de seis pessoas:

```
gosta(danilo, ana).
gosta(ana, danilo).
gosta(everton, beatriz).
gosta(fernando, beatriz).
gosta(beatriz, fernando).
gosta(beatriz, danilo).
gosta(everton, ana).
gosta(beatriz, everton).
gosta(carla, everton).
gosta(everton, carla).
```



Nesta questão você deve escrever em Prolog o predicado `santantonio(Solteiros, Casais)` que seja verdade quando:

- `Casais` for uma lista de predicados `par(X, Y)` tal que `gosta(X, Y)` e `gosta(Y, X)`;
- cada elemento `X` em `Solteiros` aparece em somente um par em `Casais`.

Por exemplo, usando as relações na figura acima, teríamos [1]:

```
?- santantonio([ana, beatriz, carla, danilo, everton, fernando], Casais).
Casais = [par(danilo, ana), par(fernando, beatriz), par(carla, everton)] ;
...

?- santantonio([ana, beatriz, carla, danilo, everton, fernando],
    [par(carla, H1), par(beatriz, H2), par(ana, H3)]).
H1 = everton,
H2 = fernando,
H3 = danilo ;
false.
```

Você pode utilizar predicados que já estão definidos na biblioteca padrão de Prolog, como `member(N, L)`, `select(E, L, LL)` e `append(L1, L2, LL)`.

[1] O seu predicado `santantonio` deve ser geral o suficiente para lidar com outras relações diferentes daquelas vistas na figura

5. Existem duas formas básicas de polimorfismo em linguagens de programação: polimorfismo universal, e polimorfismo ad-hoc. O polimorfismo universal é caracterizado por nomes que podem possuir uma gama infinita de tipos diferentes. Esta forma de polimorfismo se subdivide em duas categorias: o polimorfismo de subtipagem, e o polimorfismo paramétrico. Este último tipo de polimorfismo é extremamente útil para podermos construir módulos realmente reutilizáveis.

   (a) (3 pontos) Dê um exemplo de polimorfismo paramétrico em SML.

   (b) (3 pontos) Dê um exemplo de polimorfismo paramétrico em Java.

   (c) (4 pontos) Python é uma linguagem que não possui este tipo de polimorfismo. Porém, isto não é uma limitação de Python, pois, dadas as características do sistema de tipos da linguagem o polimorfismo paramétrico simplesmente não é necessário. Explique o por quê.

6. Considere o programa abaixo, escrito em Python. Este programa processa expressões aritméticas envolvendo somas, multiplicações e números inteiros simples:

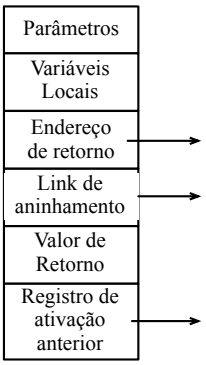
```
class Exp:
  def eval(self): return None
class Num(Exp):
  def __init__(self, num):
    self.num = num
  def eval(self): return self.num
class Plus(Exp):
  def __init__(self, e1, e2):
    self.left = e1
    self.right = e2
  def eval(self): return self.left.eval() + self.right.eval()
class Mul(Exp):
  def __init__(self, e1, e2):
    self.left = e1
    self.right = e2
  def eval(self): return self.left.eval() * self.right.eval()

print (Mul(Num(2), Plus(Num(3), Num(1))))
$> 8

print (Plus(Mul(Num(2), Num(2)), Mul(Num(3), Num(2))))
$> 10
```

Nesta questão você deve implementar esse mesmo programa em SML. Note que você terá de fazer profundas adaptações em seu programa. Em particular, SML não possui o conceito de classes, objetos e métodos.

(a) (5 pontos) Represente as expressões usando um tipo algébrico `Exp` que possua um rótulo diferente (`Num`, `Plus` e `Mul`) para casa possível tipo de expressão.

(b) (5 pontos) Implemente uma função `eval`, de tipo `Exp -> int`, que receba uma expressão aritmética e retorne o seu significado. O significado de uma expressão aritmética é o número inteiro que ela descreve. Por exemplo, `eval (Mul (Num 3, Plus (Num 2, Num 3)))` deve retornar o valor `15`.

7. Os registros de ativação guardam as informações necessárias para a execução de funções. Um registro de ativação típico é mostrado logo abaixo.



(a) (3 pontos) Normalmente, a mesma memória pode armazenar tanto as instruções quanto os dados manipulados por um programa. Porém, em geral o sistema operacional separa código e dados em segmentos de memória diferentes. Na figura logo acima, os três campos do registro de ativação de onde partem setas são ponteiros. Quais destes ponteiros apontam para dados, e quais apontam para código?

(b) (7 pontos) Considere a função `foo` logo baixo:

```
fun foo n =
  let
    fun bar 0 = []
      | bar m = n :: bar (m - 1)
  in
    bar n
  end
```

Desenhe os registros de ativação para a chamada da função `foo 3` no momento que a última chamada da função `bar` está prestes a retornar.