Prova Final de Linguagens de Programação
- DCC024B -
Ciência da Computação

Ao concordar participar da prova, você dá sua palavra de honra que suas respostas são fruto único do seu trabalho. Você pode consultar a internet, por exemplo, mas não pode consultar outros seres vivos para fazer a prova.

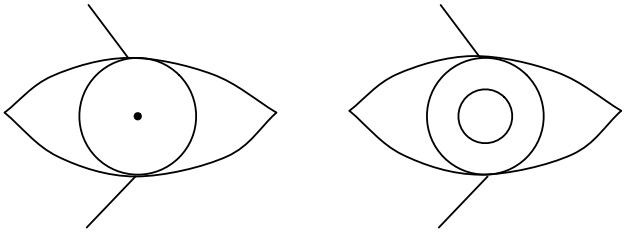
As regras do jogo:

- Você pode consultar entidades inanimadas, mas não pode consultar entidades animadas.
- Você precisa enviar três arquivos para `fernando@dcc.ufmg.br` até as 15h00 do dia 28 de Novembro, a saber:
  - `sol.sml`: solução da questão 1.
  - `sol.py`: solução da questão 2.
  - `sol.pl`: solução da questão 3.
- Seu e-mail deve ter o título `Prova_DCC024`. Não use qualquer outro título.
- O único texto no corpo de seu e-mail deve ser seu nome, e, possivelmente, a resposta da questão extra. Não escreva o código dos arquivos no e-mail: envie os arquivos como anexos.
- Caso você queira reenviar suas respostas, simplesmente responda a mensagem de seu e-mail anterior. Não envie e-mails separados!
- Não há como tirar dúvidas durante a prova. Lembre-se da cláusula sobre entidades animadas. Isso inclui escrever e-mails para o instrutor.
- Seja honesto e lembre-se: **você deu sua palavra de honra**.

Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Extra 0.5

Questão extra (0.5): a figura abaixo representa uma situação extrema. Que situação é esta?



1. Considere a seguinte definição de números naturais, implementada como um tipo algébrico em SML:

```
datatype nat = ZERO | SUCC of nat
```

   (a) (3 Pontos) Escreva uma função `nat2int` em SML, cujo tipo seja `nat -> int`. Essa função transforma um número natural em um inteiro em SML:

```
- nat2int ZERO;
val it = 0 : int

- nat2int (SUCC (SUCC ZERO));
val it = 2 : int
```

   (b) (3 Pontos) Implemente uma função `plus`, cujo tipo é: `nat -> nat -> nat`. Essa função recebe dois naturais, e retorna o natural equivalente à soma das entradas. Por exemplo:

```
- plus (SUCC(SUCC ZERO)) (SUCC(SUCC(SUCC ZERO)));
val it = SUCC (SUCC (SUCC (SUCC (SUCC ZERO)))) : nat

- plus ZERO (SUCC(SUCC(SUCC ZERO)));
val it = SUCC (SUCC (SUCC ZERO)) : nat

- plus (SUCC(SUCC(SUCC ZERO))) ZERO;
val it = SUCC (SUCC (SUCC ZERO)) : nat

- plus (plus (SUCC ZERO) (SUCC ZERO)) (SUCC ZERO);
val it = SUCC (SUCC (SUCC ZERO)) : nat
```

   Ps. você precisa operar com o tipo `nat` diretamente. Não é permitido converter naturais para inteiros, operar com os inteiros, e então fazer a conversão de inteiros para naturais.

   (c) (4 Pontos) Implemente uma função `mul`, cujo tipo é: `nat -> nat -> nat`. Essa função recebe dois naturais, e retorna o natural equivalente ao produto das entradas. Por exemplo:

```
- nat2int (mul (SUCC (SUCC ZERO)) (SUCC (SUCC (SUCC ZERO))));
val it = 6 : int

- nat2int (mul (SUCC (SUCC ZERO)) (plus (SUCC ZERO) (SUCC ZERO)));
val it = 4 : int
```

2. Nesta questão você deverá utilizar a mesma definição de números naturais vista anteriormente, mas desta vez, nós a faremos em Python, a partir das seguintes classes[1]:

```
class Zero:
  def __init__(self):
    pass

class Succ:
  def __init__(self, n):
    self.num = n
```

(a) (3 Pontos) Implemente a função `nat2int`, que transforma um número natural em um inteiro:

```
>>> nat2int(Zero())
0
>>> nat2int(Succ(Zero()))
1
```

(b) (3 Pontos) Adicione um método `__str__(self)`, a ambas as classes. Esse método deve imprimir um número natural no formato `s...sz`, havendo um número de caracteres `s` igual ao valor do número:

```
>>> n1 = Succ(Succ(Zero()))
>>> print n1
ssz
>>> n2 = Zero()
>>> print n2
z
>>> n3 = Succ(Succ(Succ(Succ(Zero()))))
>>> print n1, n2, n3
ssz z ssssz
```

(c) (4 Pontos) Implemente uma função `repeated(L)`, que recebe uma lista `L`, e retorne `True` caso haja dois números naturais iguais em `L`. Caso haja referências para objetos que não sejam números naturais, então sua função deve produzir algum tipo de erro. Por exemplo:

```
>>> execfile('sol.py')
>>> n0 = Succ(Zero())
>>> n1 = Succ(Succ(Zero()))
>>> L0 = [n0, n1, n0]
>>> repeated(L0)
True
>>> L1 = [n0, n1, Succ(Zero())]
>>> repeated(L1)
True
>>> L2 = [n0, n1, 2]
>>> repeated(L2)
Traceback (most recent call last):
# Comentário: houve um erro, log, qualquer coisa pode ser impressa aqui.
```

[1] Fique à vontade para adicionar quaisquer métodos auxiliares às classes `Zero` e `Succ`.

3. A noção de número natural pode ser escrita em Prolog facilmente:

```
num(zero).
num(succ(N)) :- num(N).
```

(a) (3 Pontos) Implemente o predicado `repeated(L)`. Este predicado é verdadeiro se `L` for uma lista que contenha dois números iguais:

```
?- repeated([zero, succ(zero), zero]).
true ;
true ;
false.

?- repeated([1, 2, 1]).
false.

?- repeated([zero, succ(zero)]).
false.

?- repeated([zero, zero]).
true ;
true ;
false.
```

(b) (3 Pontos) Implemente o predicado `plus(N0, N1, N2)`, que seja verdade caso `N2` seja o natural correspondente a soma de `N0` e `N1`. Por exemplo:

```
?- plus(zero, succ(zero), N).
N = succ(zero).

?- plus(succ(zero), succ(zero), N).
N = succ(succ(zero)).

?- plus(succ(zero), N, succ(succ(succ(zero)))).
N = succ(succ(zero)).
```

(c) (4 Pontos) Implemente um predicado `all_sums(N, L)`, que seja verdade se `L` for a lista de pares `(N0, N1)`, sendo que a soma de `N0` e `N1` é `N`. Por exemplo:

```
?- all_sums(succ(succ(zero)), L).
L = [ (zero, succ(succ(zero))), (succ(zero), succ(zero)), (succ(succ(zero)), zero)].

?- all_sums(succ(succ(succ(zero))), L).
L = [ (zero, succ(succ(succ(zero)))), (succ(zero), succ(succ(zero))),
      (succ(succ(zero)), succ(zero)), (succ(succ(succ(zero))), zero)].

?- all_sums(succ(zero), L).
L = [ (zero, succ(zero)), (succ(zero), zero)].

?- all_sums(zero, L).
L = [ (zero, zero)].
```