TÉCNICO LISBOA

Fundamentos da Programação

Solução do exame

12 de Janeiro de 2018

09:00–11:00

1. Indique se cada uma das seguintes afirmações é verdadeira ou falsa. No caso de ser falsa, justifique de forma sucinta.

   (a) **(0.5)** A abstração procedimental consiste em concentrar-nos em como a função realiza o seu trabalho.

   Resposta:

   Falsa. A abstração procedimental não considera *como* a função realiza o seu trabalho, mas sim *no que* a função faz.

   (b) **(0.5)** A execução de um programa define um processo computacional.

   Resposta:

   Verdadeira.

   (c) **(0.5)** No uso de tipos abstractos de dados (TADs) o programador tem que conhecer a representação interna dos elementos do tipo.

   Resposta:

   Falsa. programador pode ou não conhecer a representação interna, mas não a pode utilizar.

2. **(1.0)** Escreva a função `h_m_s` que recebe um inteiro positivo correspondente a um número de segundos e devolve um tuplo com três inteiros, o primeiro elemento corresponde ao número de horas contidas no número de segundos, o segundo elemento corresponde ao número de minutos corresponde ao número de minutos contido no número de segundos, removidas as horas, e o terceiro elemento ao número de segundos restantes. A sua função deve verificar a validade do argumento. Por exemplo,

```
>>> h_m_s(62)
(0, 1, 2)
>>> h_m_s(3745)
(1, 2, 25)
>>> h_m_s(3.5)
ValueError: h_m_s: o argumento deve ser inteiro positivo
```

   Resposta:

```
def h_m_s(seg):
    if isinstance(seg, int) and seg >= 0:
        horas = seg // 3600
        segs_restantes = seg % 3600
        minutos = segs_restantes // 60
        segundos = segs_restantes % 60
        return (horas, minutos, segundos)
    else:
        raise ValueError('h_m_s: o argumento deve ser inteiro positivo')
```

3. **(1.0)** Escreva a função `codifica` que recebe um número inteiro positivo e que calcula uma codificação para esse número do seguinte modo: (a) cada algarismo par é substituído pelo par seguinte, entendendo-se que o par seguinte a 8 é o 0; (b) cada algarismo ímpar é substituído pelo ímpar anterior, entendendo-se que o ímpar anterior a 1 é o 9. Não pode recorrer a cadeias de caracteres. Não é necessário verificar a correcção do argumento. Por exemplo,

```
>>> codifica(269)
487
>>> codifica(181)
909
```

Resposta:

```
def codifica(num):
    res = 0
    pos = 0
    while num != 0:
        alg = num % 10
        num = num // 10
        if alg % 2 == 0: # o algarismo é par
            alg_mod = (alg + 2) % 10
        else:
            alg_mod = (alg - 2) % 10
        res = res + alg_mod * (10 ** pos)
        pos = pos + 1

    return res
```

4. **(1.0)** Uma chave do euromilhões é constituída por cinco inteiros ordenados diferentes, entre 1 e 50 e por dois números diferentes, também ordenados, entre 1 e 12. Escreva uma função sem argumentos que devolve aleatoriamente uma lista contendo duas listas, cada uma delas contendo os constituintes de uma chave do euromilhões. Para a geração de números aleatórios utilize a função `random()`, existente na biblioteca `random`, que devolve aleatoriamente um real no intervalo $[0, 1[$. Por exemplo,

```
>>> euromilhoes()
[[23, 26, 32, 37, 48], [4, 11]]
```

Resposta:

```
def euromilhoes():

    from random import random

    def insere_ord(n, ln):
        if ln == []:
            return [n]
        else:
            i = 0
            while i < len(ln) and ln[i] < n:
                i = i + 1
            return ln[:i] + [n] + ln[i:]

    def geraLista(max_val, comp):
        nums = []
        while len(nums) != comp:
            n = int(max_val * random()) + 1
            if n not in nums:
                nums = insere_ord(n, nums)
        return nums

    return [geraLista(50, 5), geraLista(12, 2)]
```

5. (a) **(1.0)** Um número inteiro, n, diz-se *triangular* se existir um inteiro m tal que $n = 1+2+\ldots+(m-1)+m$. Escreva a função `triangular` que recebe um número inteiro positivo `n`, e cujo valor é `True` apenas se o número for triangular. No caso de `n` ser `0` deverá devolver `False`. Não é necessário verificar a correcção do argumento. Por exemplo,

```
>>> triangular(6)
True
>>> triangular(8)
False
>>> triangular(1)
False
```

Resposta:

```
def triangular (n):
    soma = 0
    i = 1
    while i < n:
        soma = soma + i
        i = i + 1
        if soma == n:
            return True
    return False
```

(b) **(1.0)** Escreva a função `nesimo_triangular` que recebe um inteiro positivo n e devolve o n-ésimo número triangular (por exemplo, se n é 3, a função devolve o terceiro número triangular). Não é necessário verificar a correcção do argumento. Por exemplo,

```
>>> nesimo_triangular(1)
3
>>> nesimo_triangular(5)
21
```

Resposta:

```
def nesimo_triangular(n):
    cont = 0
    num = 1
    while cont != n:
        if triangular(num):
            cont = cont + 1
        num = num + 1
    return num - 1
```

6. Considere a gramática em notação BNF, em que o símbolo inicial é ⟨idt⟩:

⟨idt⟩ ::= ⟨letras⟩ ⟨numeros⟩

⟨letras⟩ ::= ⟨letra⟩ |
⟨letra⟩ ⟨letras⟩

⟨numeros⟩ ::= ⟨num⟩ |
⟨num⟩ ⟨numeros⟩

⟨letra⟩ ::= `A | B | C | D`

⟨num⟩ ::= `1 | 2 | 3 | 4`

(a) **(0.5)** Diga quais são os símbolos terminais e os símbolos não terminais desta gramática.
Símbolos terminais:
Resposta:
`A` `B` `C` `D` `1` `2` `3` `4` Símbolos não terminais:
Resposta:
⟨idt⟩ ⟨letras⟩ ⟨numeros⟩ ⟨letra⟩ ⟨num⟩

(b) **(0.5)** Defina informalmente as frases geradas por esta gramática.
Resposta:
As frases começam por um número arbitrários das letras `A`, `B`, `C`, `D`, tendo pelo menos uma destas letras, seguidas por um número arbitrário dos dígitos `1`, `2`, `3`, `4`, tendo pelo menos um destes dígitos.

(c) **(1.0)** Escreva a função `reconhece`, que recebe como argumento uma cadeia de carateres e devolve *verdadeiro* se o seu argumento corresponde a uma frase da linguagem definida pela gramática e *falso* em caso contrário. Por exemplo,

```
>>> reconhece('A1')
True
>>> reconhece('ABBBBCDDDD23311')
True
>>> reconhece('ABC12C')
False
```

Resposta:

```
def reconhece(frase):
    i = 0
    while i < len(frase)-1 and frase[i] in 'ABCD':
            i = i + 1
    if i > 0: # foi encontrada pelo menos uma letra
        while i <= len(frase)-1 and frase[i] in '1234':
            i = i + 1
```

```
            return i == len(frase)
        else:
            return False
```

7. Um método básico para codificar um texto corresponde a isolar os caracteres nas posições pares para um lado e os caracteres nas posições ímpares para outro, juntando depois as duas partes anteriormente obtidas. Por exemplo, o texto `abcde` é codificado por `acebd`.

   (a) **(1.0)** Defina uma função que codifica uma cadeia de caracteres de acordo com o algoritmo apresentado. Não é necessário validar os dados de entrada. Por exemplo,

   ```
   >>> codifica('abcde')
   'acebd'
   ```

   Resposta:

   ```
   def codifica(texto):
       pares = ''
       impares = ''

       for i in range(len(texto)):
           if i % 2 == 0:
               pares = pares + texto[i]
           else:
               impares = impares + texto[i]

       return pares + impares
   ```

   (b) **(1.5)** Defina uma função que descodifica uma cadeia de caracteres de acordo com o algoritmo apresentado. Não é necessário validar os dados de entrada. Por exemplo,

   ```
   >>> descodifica('acebd')
   'abcde'
   ```

   Resposta:

   ```
   def descodifica(texto):
       res = ''
       pares = 0
       if len(texto) % 2 == 0:
           impares = len(texto) // 2
       else:
           impares = len(texto) // 2 + 1

       while impares < len(texto):
           res = res + texto[pares] + texto[impares]
           pares = pares + 1
           impares = impares + 1

       if len(texto) % 2 != 0:
           res = res + texto[pares]

       return res
   ```

8. Considere a função, `junta_ordenados`, que recebe dois tuplos contendo inteiros, ordenados por ordem crescente, e devolve um tuplo também ordenado com os elementos dos dois tuplos. Esta função não necessita de verificar a correcção dos dados de entrada. Por exemplo,

```
>>> junta_ordenados((2, 34, 200, 210), (1, 23))
(1, 2, 23, 34, 200, 210)
```

   (a) **(1.0)** Escreva a função, `junta_ordenados` usando iteração linear.
   Resposta:

```
def junta_ordenados(t1, t2):
    i1 = 0
    i2 = 0
    res = ()
    while i1 < len(t1) and i2 < len(t2):
        if t1[i1] < t2[i2]:
            res = res + (t1[i1], )
            i1 = i1 + 1
        else:
            res = res + (t2[i2], )
            i2 = i2 + 1
    res = res + t1[i1:] + t2[i2:]
    return res
```

   (b) **(1.0)** Escreva a função, `junta_ordenados` usando recursão com operações adiadas.
   Resposta:

```
def junta_ordenados(t1, t2):
    if t1 == ():
        return t2
    elif t2 == ():
        return t1
    elif t1[0] < t2[0]:
        return (t1[0],) + junta_ordenados(t1[1:], t2)
    else:
        return (t2[0],) + junta_ordenados(t1, t2[1:])
```

9. **(1.0)** Escreva a função `cria_lista_multiplos` que recebe um número inteiro positivo, e devolve uma lista com os dez primeiros múltiplos desse número. Esta função não necessita de verificar a correcção dos dados de entrada. Considere que zero é múltiplo de todos os números. Por exemplo,

```
>>> cria_lista_multiplos(6)
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54]
```

   Resposta:

```
def cria_lista_multiplos(n):
    res = []
    for i in range(10):
        res = res + [i*n]
    return res
```

10. **(1.0)** Utilizando alguns (ou todos os) funcionais sobre listas, `(transforma` ⟨função⟩`,` ⟨lista⟩`)`, `(filtra` ⟨predicado⟩`,` ⟨lista⟩`)` e `(acumula` ⟨função⟩`,` ⟨lista⟩`)`, escreva a função `todos_lista` que recebe uma lista e um predicado unário, e devolve verdadeiro caso todos os elementos da lista satisfaçam o predicado e falso em caso contrário. O corpo da sua função apenas pode ter uma instrução `return`. Por exemplo,

```
>>> todos_lista([4, 5, 6], lambda x: x > 5)
False
>>> todos_lista([4, 5, 6], lambda x: x >= 4)
True
```

Resposta:

```
def todos_lista(lst, p):
    return acumula(lambda x, y: x and y, transforma(p, lst))
```

11. **(1.0)** Escreva uma função em Python que recebe um dicionário cujos valores associados às chaves correspondem a listas de inteiros e que devolve o dicionário que se obtém "invertendo" o dicionário recebido, no qual as chaves são os inteiros que correspondem aos valores do dicionário original e os valores são as chaves do dicionário original às quais os valores estão associados. Por exemplo,

```
>>> inverte_dic({'a': [1, 2], 'b': [1, 5], 'c': [9], 'd': [4]})
{1: ['a', 'b'], 2: ['a'], 4: ['d'], 5: ['b'], 9: ['c']}
```

Resposta:

```
def inverte_dic(d):
    res = {}
    for e in d:
        for v in d[e]:
            if v in res:
                res[v] = res[v] + [e]
            else:
                res[v] = [e]
    return res
```

12. Suponha que desejava criar o tipo vetor em Python. Um vetor num referencial cartesiano pode ser representado pelas coordenadas da sua extremidade (x, y), estando a sua origem no ponto $(0, 0)$. Podemos considerar as seguintes operações básicas para vetores:

    - *Construtor*:
      $vetor : real \times real \mapsto vetor$
      $vetor(x, y)$ tem como valor o vetor cuja extremidade é o ponto (x, y).
    - *Seletores*:
      $abcissa : vetor \mapsto real$
      $abcissa(v)$ tem como valor a abcissa da extremidade do vetor v.
      $ordenada : vetor \mapsto real$
      $ordenada(v)$ tem como valor a ordenada da extremidade do vetor v.

- *Reconhecedores*:
  $eh\_vetor : universal \mapsto lógico$
  $eh\_vetor(arg)$ tem valor $verdadeiro$ apenas se arg é um vetor.
  $eh\_vetor\_nulo : vetor \mapsto lógico$
  $eh\_vetor\_nulo(v)$ tem valor $verdadeiro$ apenas se v é o vetor (0, 0).
- *Teste*:
  $vetores\_iguais : vetor \times vetor \mapsto lógico$
  $vetores\_iguais(v_1, v_2)$ tem valor verdadeiro apenas se os vetores v_1 e v_2 são iguais.

(a) **(0.5)** Defina uma representação para vetores.

(b) **(1.0)** Escreva as operações básicas, de acordo com a representação escolhida.

(c) **(1.0)** Escreva a função `soma_vetores` que soma os dois vetores que são seus argumentos. A soma dos vetores representados pelos pontos (a, b) e (c, d) é dada pelo vetor $(a + c, b + d)$.

13. **(1.5)** Defina uma classe que corresponde a uma urna de uma votação. A sua classe deve receber a lista dos possíveis candidatos (uma lista de cadeias de caracteres) e manter como estado interno o número de votos em cada candidato. Esta classe pode receber um voto num dos possíveis candidatos, aumentando o número de votos nesse candidato em um. Deve também permitir apresentar os resultados da votação. Não é necessário validar os argumentos.

Resposta:

```
class votacao:

    def __init__(self, candidatos):
        self.cand = {}
        self.nulos = 0
        for c in candidatos:
            self.cand[c] = 0

    def vota(self, c):
        if c in self.cand:
            self.cand[c] = self.cand[c] + 1
        else:
            self.nulos = self.nulos + 1

    def resultados(self):
        maior = 0
        for c in self.cand:
            if self.cand[c] > maior:
                maior = self.cand[c]
        for c in self.cand:
            print(c, 'votos ', self.cand[c])
        vencedor = []
        for c in self.cand:
            if self.cand[c] == maior:
                vencedor = vencedor + [c]
        if len(vencedor) == 1:
            print('Candidato vencedor ', vencedor[0])
        else:
```

```
            print('Candidatos empatados')
            for c in vencedor:
                print(' ', c)
        if self.nulos != 0:
            print(self.nulos, ' votos nulos')
```