TÉCNICO LISBOA

FUNDAMENTOS DA PROGRAMAÇÃO

2º exame, 30 de janeiro de 2015

Duração: 2h

Nome: _____ Número:_____

- Esta prova, individual e sem consulta, tem 10 páginas com 9 perguntas. A cotação de cada pergunta está assinalada entre parêntesis.

- Escreva o seu número em todas as folhas da prova. O tamanho das respostas deve ser limitado ao espaço fornecido para cada questão. O corpo docente reserva-se o direito de não considerar a parte das respostas que excedam o espaço indicado.

- Pode responder usando lápis.

- Em cima da mesa devem apenas estar o enunciado, caneta ou lápis e borracha e cartão de aluno. Não é permitida a utilização de folhas de rascunho, telemóveis, calculadoras, etc.

1. **(1.0+1.0)** Suponha que se pretendem somar os *n* primeiros números quadrados, 1+4+9+16+...+n^2 onde é *n* é um parâmetro.
    a. Usando programação imperativa, escreva uma função que calcule e devolva o valor da soma.

Solução:

```
def soma_dos_quadrados(n):
     ''' soma_dos_quadrados : inteiro -> inteiro
         soma_dos_quadrados(n) devolve a soma dos termos
         i*i para i = 1 ate n. '''

     soma = 0
     for k in range(1,n+1):
          soma += k * k
     return soma
```

    b. Usando programação funcional, sem ciclos nem atribuições explícitas, escreva uma função que calcule e devolva o valor da soma.

Solução:

```
def soma_dos_quadrados(n):
     ''' soma_dos_quadrados : inteiro -> inteiro
         soma_dos_quadrados(n) devolve a soma dos termos
         i*i para i = 1 ate n. '''
     if n < 1:
          return 0
     else:
          return n * n + soma_dos_quadrados(n-1)
```

2. **(2.0)** Escreva uma função `inteiro_valido` que recebe como argumento uma cadeia de caracteres e devolve `True` caso esta corresponda a um número inteiro não negativo válido, ou seja um número sem parte decimal, e `False` em caso contrário. Por exemplo, deverá ser possível obter a seguinte interacção:

```
>>> inteiro_valido('abc')
False
>>> inteiro_valido('123.4')
False
>>> inteiro_valido('123')
True
>>> inteiro_valido('-123')
False
>>> inteiro_valido('5.0')
False
```

Solução:

```
def inteiro_valido(s):
     ''' inteiro_valido : cad. caracteres -> logico
         inteiro_valido(s) devolve True caso a string s
         corresponda a um inteiro valido, e False
         em caso contrario.'''

     for c in s:
          if c < '0' or c > '9':
               return False

     return True
```

3. **(2.0)** Usando a função da alínea anterior, escreva uma função calculadora que pede repetidamente ao utilizador para introduzir dois números inteiros e um dos quatro operadores +, –, * e /, e efectua o cálculo associado. A função deve terminar quando o utilizador introduzir entradas inválidas. Por exemplo, deverá ser possível obter obter a seguinte interacção:

```
>>> calculadora()
Introduza um inteiro: 12
Introduza um inteiro: 34
Introduza um operador aritmetico (+, -, *, /): +
12 + 34 = 46
Introduza um inteiro: 123
Introduza um inteiro: 456
Introduza um operador aritmetico (+, -, *, /): /
123 / 456 = 0.26973684210526316
Introduza um inteiro: abc
Introduza um inteiro: def
Introduza um operador aritmetico (+, -, *, /): hij
Fim.
```

Solução:

```
def calculadora():
     ''' calculadora : {} -> {}
         calculadora() permite ao utilizador efectuar
         uma sequencia de calculos simples.'''

     terminar = False

     while not terminar:

          num1 = input('Introduza um inteiro: ')
          num2 = input('Introduza um inteiro: ')
          oper = input('Introduza um operador (+, -, *, /): ')

          if not inteiro_valido(num1) or \
              not inteiro_valido(num2) or \
              oper not in ('+', '-', '*', '/'):

               terminar = True

          else:
               print(num1, oper, num2, '=', eval(num1+oper+num2))

     print('Fim.')
```

4. **(3.0)** Escreva a função `tabuleiro_terminado`, pedida no projecto, que recebe como argumento um elemento do tipo tabuleiro e devolve `True` caso este corresponda a um tabuleiro terminado de 2048. Um tabuleiro considera-se terminado se não tiver casas vazias nem casas adjacentes com o mesmo número. Na sua solução,

a. deve considerer que a função `tabuleiro_terminado` **não é** operação básica do tipo *tabuleiro* (ao contrário do que acontecia no projecto).
b. *o tabuleiro não deverá percorrido mais do que uma vez* (pela função ou por outras funções auxiliares).

Pode assumir como definidas as seguintes operações básicas dos tipos *tabuleiro* e *coordenada*:

- `cria_coordenada : int × int → coordenada`
- `coordenada_linha : coordenada → int`
- `coordenada_coluna : coordenada → int`
- `cria_tabuleiro : {} → tabuleiro`
- `tabuleiro_posicao : tabuleiro × coordenada → int`
- `tabuleiro_pontuacao : tabuleiro → int`

Solução:

```
def tabuleiro_terminado(tab):
    ''' tabuleiro_terminado : tabuleiro -> logico
        tabuleiro_terminado(t) devolve True caso t corresponda
        a um tabuleiro terminado, e False em caso contrario.'''
    for l in range(4):
        for c in range(4):
            if tabuleiro_posicao(t, cria_coordenada(l+1, c+1)) == 0 or \
              (l < 3 and tabuleiro_posicao(t, cria_coordenada(l+1, c+1)) \
              == tabuleiro_posicao(t, cria_coordenada(l+2, c+1))) or \
              (c < 3 and tabuleiro_posicao(t, cria_coordenada(l+1, c+1)) \
              == tabuleiro_posicao(t, cria_coordenada(l+1, c+2))):

                  return False

    return True
```

5. **(1.0+1.0)** Considere a seguinte função recursiva, que recebe dois inteiros, *n* e *k*, e um dígito, *d*, e altera o dígito de *n* na posição *k* para *d*.

```
def altera_digito(n, k, d):
    ''' altera_digito : int x int x int -> int
        altera_digito(n, k, d) altera o digito de n na
        posicao k para d.'''
    if k == 1:
        return (n // 10) * 10 + d
    else:
        return altera_digito(n // 10, k - 1, d) * 10 + n % 10
```

   a. Qual dos 3 parâmetros, *n*, *k*, ou *d*, é que tem maior impacto no tempo de execução da função?

Solução: O parâmetro *k* determina o número de chamadas recursivas que são realizadas pela função, pelo que é o parâmetro que influencia de forma mais determinante o tempo de execução.

   b. Como é que esse tempo de execução depende desse parâmetro?

Solução: O tempo de execução cresce linearmente com o valor de *k*.

6. **(1.0+1.0)** Um número diz-se perfeito se for igual à soma dos seus divisores, sem contar com o divisor que é igual ao próprio número. Os dois primeiros números perfeitos são 6=1+2+3 e 28=1+2+4+7+14.
    a. Escreva em Python uma função, `lista_divisores`, que aceita um número inteiro positivo e devolve uma lista com todos os divisores desse número, inferiores ao próprio número (incluindo o número 1, que divide todos os números)

Solução:

```
def lista_divisores(n):
      ''' lista_divisores : int -> lista
          lista_divisores(n) devolve uma lista contendo todos os
          divisores de n.'''

      divisores = [1]

      for i in range(2, n):
           if n % i == 0:
                divisores = divisores + [i]

      return divisores
```

    b. Usando a função anterior, escreva uma função, `numero_perfeito`, que aceita um número inteiro positivo e devolve `True` se ele for perfeito, e `False` se não for perfeito.

Solução:

```
def numero_perfeito(n):
      ''' numero_perfeito : int -> logico
          numero_perfeito(n) devolve True caso n seja um numero
          perfeito e False em caso contrario.'''

      divisores = lista_divisores(n)
      soma = 0

      for d in divisores:
           soma = soma + d

      return soma == n
```

7. **(2.0)** Uma matriz é dita esparsa (ou rarefeita) quando a maior parte dos seus elementos é zero. As matrizes esparsas aparecem em grande número de aplicações em engenharia. Uma matriz esparsa pode ser representada por um dicionário que contém um par com a posição de um elemento na matriz (linha e coluna) e o respectivo valor. Por exemplo, nesta representação a matriz

   0 7 0 0
   0 0 1 0
   0 0 0 0
   0 0 9 0

   seria representada pelo dicionário {(1,2):7, (2,3):1, (4,3):9}

   Escreva em Python uma função `escreve_esparsa` que aceite o número de linhas da matriz, o número de colunas da matriz e uma dicionário de acordo com o formato do exemplo e escreva a matriz por extenso, também como no exemplo.

   Solução:

```
def escreve_esparsa(nl, nc, m):
      ''' escreve_esparsa : int x int x dic -> {}
          escreve_esparsa(l, c, m) escreve para o ecra a matriz m
          com l linhas e c colunas.'''

      for l in range(nl):
            linha = ''
            for c in range(nc):
                  if (l+1, c+1) in m:
                        linha = linha + str(m[(l+1, c+1)]) + ' '
                  else:
                        linha = linha + '0 '

            print(linha)
```

8. **(1.0+1.5)** Suponha que desejava criar o tipo paciente num sistema de informação. Um paciente deverá ser representado pelo nome e ano de nascimento.
    a. Defina uma representação e as operações básicas para o tipo paciente, em Python.

Solução:

```
def cria_paciente(nome, ano):
      ''' cria_paciente : {} -> paciente
          cria_paciente() devolve um novo paciente'''

      return {'nome': nome, 'ano_de_nascimento': ano}

def paciente_nome(paciente):
      ''' paciente_nome : paciente -> cad. caracteres
          paciente_nome(p) retorna o nome do paciente p. '''

      return p['nome']

def paciente_ano_de_nascimento(paciente):
      ''' paciente_nome : paciente -> inteiro
          paciente_nome(p) retorna o ano de nascimento do
          paciente p. '''

      return p['ano_de_nascimento']


def eh_paciente(x):
      ''' eh_paciente : universal -> lógico
          eh_paciente(x) retorna True sse x for um paciente. '''

      return isinstance(x, dict) and len(x) == 2 and \
          'nome' in x and 'ano_de_nascimento' in x and \
          isinstance(x['nome'], str) and \
          isinstance(x['ano_de_nascimento'], int)


def paciente_igual(p, q):
      ''' paciente_igual : paciente × paciente -> lógico
          paciente_igual(q, q) retorna True se os pacientes p e q
          forem iguais. '''

      return p['nome'] == q['nome'] and \
          p['ano_de_nascimento'] == q['nao_de_nascimento']
```

    b. Suponha que dado o tipo paciente queremos implementar o tipo fila_de_espera. O tipo fila_de_espera deverá suportar as seguintes operações:

```
cria_fila_de_espera : None -> fila_de_espera
colocar_em_espera : fila_de_espera x paciente -> None
proximo_paciente : fila_de_espera -> paciente
retirar_proximo : fila_de_espera -> fila_de_espera
```

Escreva em Pyhton estas operações, definindo uma representação para o tipo fila_de_espera.

Solução:

```
def cria_fila_de_espera():
    ''' cria_fila_de_espera : {} -> fila_de_espera
        cria_fila_de_espera() devolve uma nova fila de espera
        vazia.'''

    return []

def colocar_em_espera(fila, paciente):
    ''' colocar_em_espera : fila x paciente -> {}
        colocar_em_espera(f, p) acrescenta o paciente p 'a fila
        de espera f.'''

    fila.append(paciente)

def proximo_paciente(fila):
    ''' proximo_paciente : fila -> paciente
        proximo_paciente(f) devolve o proximo paciente da fila f,
        caso exista, e erro em caso contrario.'''

    if len(fila) > 0:
        return fila[0]
    else:
        raise ValueError ('proximo_paciente: fila vazia.')

def retirar_proximo(fila):
    ''' retirar_paciente : fila -> {}
        retirar_paciente(f) remove o proximo paciente da fila
        f, caso exista, e da erro em caso contrario.'''

    if len(fila) > 0:
        del(fila[0])
    else:
        raise ValueError ('retirar_paciente: fila vazia.')
```

9. **(1.5+1.0)** Considere a função `seleciona_enesimo` que, dada uma lista de números de dimensão *m* e um valor para o parâmetro *n*, devolva o número que é o enésimo maior na lista. Por exemplo, devemos obter a seguinte interacção:

```
>>> lst = [3, 6, 1, 10, 17, 13, 5, 18, 55, 23]
>>> seleciona_enesimo(lst,1)
```

```
55
>>> seleciona_enesimo(lst,3)
18
>>> seleciona_enesimo(lst,5)
13
```

a. Escreva uma realização da função `seleciona_enesimo`

Solução:

```
def seleciona_enesimo(l, n):
     ''' seleciona_enesimo : lista x int -> int
         seleciona_enesimo(l, n) devolve o nesimo maior elemento
         da lista l.'''

     if n > len(l):
          raise ValueError ('seleciona_enesimo: indice invalido.')

     alterado = True
     ultimo = -1
     while alterado and ultimo >= -n:
          alterado = False

          for i in range(len(l) + ultimo):
               if l[i] > l[i + 1]:
                    l[i], l[i + 1] = l[i + 1], l[i]
                    alterado = True
          ultimo = ultimo - 1

     return l[-n]
```

b. Usando a notação O, qual a complexidade da função que realizou, como função de *m* e de *n*?

Solução: O tempo de execução cresce proporcionalmente a *mm* (ou *O(mm)*, ou *Θ(mm)). (Porém, noutras soluções, a complexidade será diferente. Se usar outro algoritmo de ordenação mais eficiente, a complexidade será O(m log m). Se selecionar elementos em sequência, primeiro o maior, depois o segundo, etc, a complexidade será O(nm))*