MAC2166 - Introdução à Computação (18/6) - 2019S1 **Avaliação 3**

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9

Utilize caneta azul ou preta e preencha completamente a quadrícula.
Exemplo: ■. **Não use ☒.**

Turma: (somente um número; consulte a pessoa responsável se não souber)

4	5	6	7	8	9	10	11	12	13	14	20

⟵ Marque as quadrículas ao lado para formar o seu número USP e escreva seu nome completo em letra legível na linha pontilhada abaixo. **Se seu número possui menos que 8 dígitos complete com zeros à esquerda.**

Nome:

.....

Esta avaliação tem duração de 120 minutos. Não desmonte este caderno.

Q1 [2 pontos] Simule o código abaixo e assinale a opção correspondente à saída impressa do programa. Ignore quebras de linhas geradas pelo comando `print`.

```
def f1 (vet, i1, i2, i) :  #
    if (i1==i2):
        print("(%d :  %d)"%(i, vet[i1]))
        return vet[i1]
    aux = vet[i1] + f1(vet, i1+1, i2, i+1)
    print("(%d :  %d) "%(i, aux))
    return aux


def f2 (vet1, vet2):
    aux1 = f1(vet1, vet2[0], vet2[1], 0)
    aux2 = f1(vet1, vet2[2], vet2[3], 0)
    print(aux1, aux2)

vet = [2, 9, 8, 4, 2, 5, 1, 0, 6]
ind = [1, 3, 3, 5]
f2(vet, ind)
```

Rascunho

- ☐ `(1:12) (2:4) (0:21) (0:11) (1:7) (2:5) 4 5`
- ☐ `(2:4) (2:12) (3:1) (2:5) (1:7) (0:11) 7 11`
- ☐ `(0:11) (0:21) (1:7) (1:12) (2:5) (2:4) 7 12`
- ☐ `(2:5) (2:4) (1:7) (1:12) (0:11) (0:21) 7 21`
- ☐ `(1:12) (0:21) (2:5) (1:7) (0:11) (2:4) 7 21`
- ☐ `(2:4) (1:12) (0:21) (2:5) (1:1) (12:21) 21 21`
- ☐ `(2:4) (1:12) (2:1) (2:5) (1:7) (0:11) 21 11`
- ☐ `(2:4) (1:12) (0:21) (7:2) (11:7) (21:0) 21 0`
- ☐ `(2:4) (1:12) (0:21) (2:5) (1:7) (0:11) 11 21`
- ☐ `(1:12) (0:21) (1:7) (0:11) (2:5) (2:4) 7 21`
- ☐ `(0:21) (1:12) (2:4) (0:11) (1:7) (2:5) 4 5`
- ☐ `(2:4) (1:12) (0:21) (2:5) (1:7) (0:11) 21 11`
- ☐ `(0:11) (1:7) (2:5) (0:21) (1:12) (2:4) 5 4`
- ☐ `(2:4) (1:12) (0:21) (2:5) (1:7) (1:11) 21 5`
- ☐ `(2:5) (1:7) (0:11) (2:4) (1:12) (0:21) 11 21`
- ☐ `(1:12) (0:21) (2:4) (0:11) (1:7) (2:5) 4 5`

Q2 [2 pontos] O foco desse exercício é conseguir uma sequência de discplinas cujos pré-requisitos já tenham sido feitos (ou não tenham sequer um). Para simplificar, suponha que cada disciplina (representada por um inteiro em lista/vetor `vet[]`) tenha no máximo um pré-requisito (um inteiro em lista/vetor `pr[]`). Se a disciplina `vet[i]` não tem pré-requisitos, então `pr[i]=-1`. Se `vet[i]` depende de `vet[j]`, então `pr[i]=j`.

Preencha as lacunas abaixo (L0 até L9), de forma a obter um programa que imprima corretamente uma sequência. Nesse algoritmo, após "cursar" a disciplina `vet[i]`, marca-se `pr[i]=-2`.

```
def buscaPR (pr, n) :
    L0
    L1 :
      if (L2 ) :
        L3
      L4
    return -1; # erro => NENHUMA sem prerequisitos

def prerequisitos (vet, pr, n) :
    for i in range(n) :
      L5
      if (L6 ) :
        print("NAO existe mais disciplinas livres");
        return -1; # erro => NENHUMA sem prerequisito
      print(ind);
      L7
      for j in range(n) :
        if (L8 ) :
          L9
    return 1;
...
resp = prerequisitos(vet, pr, N);
```

Rascunho

Para cada um dos **10** itens a seguir, correspondendo as lacunas no código acima, assinale a única resposta que torna o programa acima correto. Itens errados sofrerão descontos.

L0:	☐ `pr[0]=-1`	☐ `i = 0`	☐ `if (n==0) :return -1`	☐ `i = n`	☐ `pr[n]=-1`
L1:	☐ `i += 1`	☐ `pr[i]=-1`	☐ `while (i<n)`	☐ `return i`	☐ `if (i==n):return -1`
L2:	☐ `pr[i] == 0`	☐ `i==n`	☐ `i<n`	☐ `pr[i] == -1`	☐ `pr[n]==0`
L3:	☐ `break`	☐ `return -1`	☐ `i += 1`	☐ `return i`	☐ `i=n`
L4:	☐ `i += 1`	☐ `i=n`	☐ `break`	☐ `return i`	☐ `pr[i]=-1`
L5:	☐ `ind= buscaPR(pr,n)`	☐ `buscaPR(pr,n)`	☐ `ind=i`	☐ `ind=-1`	☐ `pr[i]= buscaPR(pr,n)`
L6:	☐ `i==ind`	☐ `ind==-1`	☐ `i==-1`	☐ `ind==-2`	☐ `i==n`
L7:	☐ `pr[i]=ind`	☐ `pr[i]=-2`	☐ `pr[ind]=-1`	☐ `pr[i]=-2`	☐ `pr[ind]=-2`
L8:	☐ `pr[j]==ind`	☐ `pr[j]==-1`	☐ `pr[ind]==i`	☐ `pr[ind]==j`	☐ `pr[j]==i`
L9:	☐ `pr[ind]=-2`	☐ `pr[ind]=-1`	☐ `pr[i]=-1`	☐ `pr[j]=-2`	☐ `pr[j]=-1`

Q3 [3 pontos] Vimos em aula que, no algoritmo de ordenação por inserção, há uma busca pela posição correta na lista, do item a ser ordenado. Essa busca é feita de forma sequencial na parte da lista que já está ordenada. Assim, seria possível trocarmos a busca sequencial por uma busca binária (chamada em L7). Preencha as lacunas no código abaixo (L1 até L8), de forma a obter um programa que realiza a ordenação por inserção utlizando a busca binária.

```
def BB(L, x, inic, fim):  # busca binaria
    if inic == fim:
        if L[inic] > x:
            return inic
        else:
            return inic+1
    if inic > fim:
        return inic
    L1
    if L[meio] < x:
        L2
    L3
        L4
    else:
        L5

def ordenacao_insercao(L):
    L6
        x = L[i]
        L7
        L8
    return L
```

Rascunho

Para cada um dos **8** itens a seguir, correspondendo as lacunas no código acima, assinale a única resposta que torna o programa acima correto. Itens errados sofrerão descontos.

L1:
- ☐ `meio=(inic+fim)%2`
- ☐ `meio=(inic+fim)//2`
- ☐ `meio=inic-1`
- ☐ `meio=fim-1`
- ☐ `meio=inic+1`

L2:
- ☐ `return BB(L,x,meio+1,fim-1)`
- ☐ `return BB(L,x,meio,fim)`
- ☐ `return BB(L,x,meio-1,fim)`
- ☐ `return BB(L,x,meio,fim-1)`
- ☐ `return BB(L,x,meio+1,fim)`

L3:
- ☐ `elif L[meio]<x:`
- ☐ `elif L[inic+fim]==meio:`
- ☐ `elif L[meio]==x:`
- ☐ `elif L[(inic+fim)//2]==meio:`
- ☐ `elif L[meio]>x:`

L4:
- ☐ `return BB(L,x,inic,meio)`
- ☐ `return BB(L,x,inic,meio-1)`
- ☐ `return BB(L,x,inic+1,meio-1)`
- ☐ `return BB(L,x,meio+1,meio-1)`
- ☐ `return BB(L,x,inic-1,meio)`

L5:
- ☐ `return meio`
- ☐ `return L[meio]`
- ☐ `return fim`
- ☐ `return L[meio/2]`
- ☐ `return inic`

L6:
- ☐ `for i in range(len(L)):`
- ☐ `while i <= len(L):`
- ☐ `for i in range(1,len(L)):`
- ☐ `while i > len(L):`
- ☐ `while i < len(L):`

L7:
- ☐ `j=BB(L,x,0,len(L))`
- ☐ `j=BB(L,x,0,i)`
- ☐ `j=BB(L,x,i,len(L))`
- ☐ `j=BB(L,x,0,i-1)`
- ☐ `j=BB(L,x,i+1,len(L))`

L8:
- ☐ `L[j]=x`
- ☐ `x=L[j]`
- ☐ `L[j+1]=x`
- ☐ `L=L[:]+L[j:i]+L[i+1:]`
- ☐ `L=L[:j]+[x] +L[j:i]+L[i+1:]`

Q4 [3 pontos] Suponha uma matriz `M[][]` de inteiros representando uma imagem, cujos valores seja a cor do *pixel* daquela posição. O objetivo é obter um código que filtre a imagem eliminando brancos: o *pixel* da posição *(i,j)* (linha,coluna) é branco $\Leftrightarrow$ `M[i][j]=0`. O filtro é feito da seguinte forma: a matriz resultante `MR[][]`, terá os mesmos valores que `M[][]`, exceto (eventualmente) nas posições em que o *pixel* era branco, nesse caso deve trocá-lo pela média aritmética (inteira) dos *pixeis* vizinhos. Por exemplo, se o elemento da linha 0, coluna 0 for branco, então na posição resultante deverá ser a média entre os *pixeis* da linha 0, coluna 1, com aquele da linha 1, coluna 0 (ou seja, some-os e divida por 2). Entretanto se o *pixel* tiver 3 vizinhos, some os 3 e divida por 3, da mesma forma, se tiver 4, some os 4 e divida-os por 4. O cabeçalho deve ser `def computaMedia(M,i,j,m,n)` e é invocada, no programa principal, por laço duplo do tipo: `for i ... for j ... MR[i][j]=computaMedia(M,i,j,m,n);`.
DICA 1: Sendo m, n o número de linhas e de colunas de `M`, pode-se supor que tanto m quanto n sejam no mínimo 2.
DICA 2: No quadro abaixo estão os trechos com seu número e indentação, como `T18-2`.

```
# T1 - 1
if (M[i][j]==0) :
 return M[i][j]
# T2 - 1
if (M[i][j]!=0) :
 return M[i][j]
# T3 - 1
if (M[i][j]!=0) :
 return M[i-1][j]
# T4 - 1
if (i==0) :
# T5 - 1
if (i!=0) :
# T6 - 1
if (i==j) :
# T7 - 2
 if (j==0 ) :
  return (M[1][0]+M[0][1])//2
# T8 - 2
 if (j==0 ) :
  return (M[0][0]+M[0][1])//2
# T9 - 2
 if (j==n-1) :
   return
      (M[0][n-2]+M[1][n-1])//2
# T10 - 2
 if (j==n-1) :
   return
      (M[0][n-1]+M[1][n-2])//2
# T11 - 2
   return (M[0][j-1]+M[0][j+1]+M[1][j])//3
# T12 - 2
   return (M[0][j]+M[0][j+1]+M[1][j])//3
# T13 - 2
 if (i==n-1) :
   return (M[0][j]+M[0][j+1]+M[1][j])//3
# T14 - 1
elif (i==m-1) :
# T15 - 1
elif (i==n-1) :
# T16 - 2
 if (j==0) :
   return (M[m-2][0]+M[m-1][1])//2
# T17 - 2
 if (j==0) :
   return (M[m-1][0]+M[m-2][1])//2
# T18 - 2
 if (j==1) :
   return (M[m-1][0]+M[m-1][0])//2
# T19 - 2
 if (j==n-1) :
   return (M[m-1][n-2]+M[m-2][n-1])//2
# T20 - 2
 if (j==n-1) :
   return (M[m-2][n-2]+M[m-2][n-2])//2
# T21 - 2
 if (j==n) :
   return (M[m-1][n-1]+M[m-1][n-1])//2
# T22 - 2
   return (M[m-1][j-1]+M[m-1][j+1]+M[m-2][j])//3
# T23 - 2
   return (M[j-1][m-1]+M[j+1][m-1]+M[j][m-2])//3
# T24 - 2
 if (i==n-1) :
   return (M[m-2][j-1]+M[m-1][j+1]+M[m-1][j])//3
# T25 - 1
if (j==0) :
# T26 - 1
if (i==0 ) :
 return (M[1][0]+M[0][1])//2
# T27 - 2
   return (M[i-1][0]+M[i][1]+M[i+1][0])//3
# T28 - 2
 if (j!=0) :
   return (M[i-1][0]+M[i][1]+M[i+1][0])//3
# T29 - 1
if (j==n-1) :
# T30 - 1
if (i==0 ) :
 return (M[0][n-2]+M[1][n-1])//2
# T31 - 2
   return (M[i-1][n-1]+M[i][n-2]+M[i+1][n-1])//3
# T32 - 2
   return (M[n-1][i-1]+M[n-2][i]+M[n-1][i+1])//3
# T33 - 1
return (M[i][j-1]+M[i-1][j]+M[i][j+1]+M[i+1][j])//4
# T34 - 1
return (M[j-1][i]+M[j][i-1]+M[j+1][i]+M[j][i+1])//4
```

Assinale a ÚNICA alternativa que contém os blocos corretos na ORDEM correta.

- ☐ `2, 4, 8, 9, 12, 14, 16, 19, 22, 27, 28, 29, 31, 33`
- ☐ `3, 4, 7, 9, 11, 14, 16, 19, 23, 25, 27, 29, 31, 34`
- ☐ `2, 4, 8, 9, 12, 14, 16, 19, 22, 25, 27, 30, 31, 33`
- ☐ `2, 5, 7, 9, 12, 14, 16, 19, 22, 25, 28, 29, 31, 33`
- ☐ `3, 4, 7, 9, 11, 14, 16, 19, 22, 26, 27, 29, 31, 34`
- ☐ `2, 5, 7, 9, 12, 14, 16, 19, 22, 25, 27, 29, 32, 33`
- ☐ `3, 4, 7, 9, 11, 14, 16, 19, 24, 26, 27, 29, 31, 33`
- ☐ `3, 4, 7, 9, 11, 15, 16, 19, 22, 26, 27, 29, 31, 34`
- ☐ `2, 4, 7,10, 11, 15, 16, 19, 22, 25, 27, 29, 31, 33`
- ☐ `2, 4, 8, 9, 11, 14, 18, 19, 22, 25, 27, 30, 31, 33`
- ☐ `1, 4, 7, 9, 12, 14, 16, 19, 22, 26, 27, 29, 31, 33`
- ☐ `3, 4, 7,10, 11, 14, 16, 19, 22, 26, 27, 29, 31, 33`
- ☐ `2, 4, 8, 9, 11, 14, 16, 19, 22, 27, 28, 29, 31, 33`
- ☐ `1, 4, 7,10, 11, 15, 17, 19, 22, 25, 27, 29, 31, 33`
- ☐ `1, 4, 8, 9, 12, 14, 16, 19, 22, 25, 27, 29, 31, 33`
- ☐ `2, 5, 7,10, 11, 15, 16, 19, 22, 25, 27, 29, 31, 33`
- ☐ `2, 4, 7, 9, 11, 14, 16, 19, 22, 25, 27, 29, 31, 33`
- ☐ `2, 5, 7, 9, 12, 14, 16, 19, 22, 25, 27, 29, 31, 34`
- ☐ `1, 4, 7, 9, 12, 16, 16, 19, 22, 25, 27, 29, 31, 33`
- ☐ `3, 4, 8, 9, 11, 14, 16, 19, 22, 25, 27, 29, 31, 34`

Rascunho