MAC2166 Introdução à Computação

Prova 3

QUESTÃO 1

Simule a execução do programa abaixo, destacando a sua saída. A saída do programa consiste de tudo que resulta dos comandos `printf`.

```
#include <stdio.h>

void f0(int k, int n, int w[6]);
int  f1(int k, int *a, int *b, int j);
int  f2(int n, int m[2][3], int v[6]);

int main() {
  int nusp, n, i, j, k, v[6], w[6], m[2][3];
  float x;

  printf("Digite o seu no. USP: ");
  scanf("%d", &nusp); /* use o SEU no. USP */
  printf("main 0: nusp=%d\n", nusp);

  n = nusp / 10;
  k = n % 10;
  printf ("main 1: n=%d k=%d\n", n, k);

  for (i = 0; i < 6; i++) {
      v[i]   = i*10 + k;
      w[5-i] = i*10 + k;
  }

  f0(2, 6, v);
  f0(3, 6, w);

  i = 2;
  j = 3;
  i = f1(4, &j, &i, v[1]);
  printf("main 5: i=%d j=%d v[1]=%d\n",
          i, j, v[1]);

  i = 3;
  j = 4;
  w[5] = f1(6, &w[j], &w[j], w[i]);
  printf("main 7: i=%d j=%d w[%d]=%d w[5]=%d\n",
          i, j, j, w[j], w[5]);

  x = f2(6, m, w) / 2;
  printf("main 8: %f\n", x);

  for (i = 0; i < 2; i++) {
    printf("main %d: ", i+9);
    for (j = 0; j < 3; j++)
      printf("%d ", m[i][j]);
    printf("\n");
  }

  return 0;
}

void f0(int k, int n, int w[6]) {
  int i;

  printf("f0   %d: ", k);
  for (i = 0; i < n; i++)
    printf("vet[%d]=%d ", i, w[i]);
  printf("\n");
}

int f1(int k, int *a, int *b, int j) {
  int i;

  i  =  0;
```

```
  j  =  j + 1;
  *a = *a + 2;
  *b = *b + 3;
  printf("f1   %d: i=%d j=%d *a=%d *b=%d\n",
         k, i, j, *a, *b);

  return 70;
}

int f2(int n, int m[2][3], int v[6]) {
  int i;

  for (i = 0; i < n; i++)
    m[i/3][i%3] = v[i];

  return 5;
}
```

SOLUÇÃO

A resposta depende do resto da divisão do segundo dígito menos significativo do seu número USP (`(nusp%100)/10`). Por exemplo, para no. USP = 1234569 o dígito 6 é o segundo menos significativo. Teste com o seu no. USP e compare a resposta.

`(nusp%100)/10 == 0`.

```
Digite o seu no. USP: 1234507
main 0: nusp=1234507
main 1: n=123450 k=0
f0   2: vet[0]=0 vet[1]=10 vet[2]=20 vet[3]=30 vet[4]=40 vet[5]=50
f0   3: vet[0]=50 vet[1]=40 vet[2]=30 vet[3]=20 vet[4]=10 vet[5]=0
f1   4: i=0 j=11 *a=5 *b=5
main 5: i=70 j=5 v[1]=10
f1   6: i=0 j=21 *a=15 *b=15
main 7: i=3 j=4 w[4]=15 w[5]=70
main 8: 2.000000
main 9: 50 40 30
main 10: 20 15 70
```

`(nusp%100)/10 == 1`.

```
Digite o seu no. USP: 1234517
main 0: nusp=1234517
main 1: n=123451 k=1
f0   2: vet[0]=1 vet[1]=11 vet[2]=21 vet[3]=31 vet[4]=41 vet[5]=51
f0   3: vet[0]=51 vet[1]=41 vet[2]=31 vet[3]=21 vet[4]=11 vet[5]=1
f1   4: i=0 j=12 *a=5 *b=5
main 5: i=70 j=5 v[1]=11
f1   6: i=0 j=22 *a=16 *b=16
main 7: i=3 j=4 w[4]=16 w[5]=70
main 8: 2.000000
main 9: 51 41 31
main 10: 21 16 70
```

`(nusp%100)/10 == 2`.

```
Digite o seu no. USP: 1234527
main 0: nusp=1234527
main 1: n=123452 k=2
f0   2: vet[0]=2 vet[1]=12 vet[2]=22 vet[3]=32 vet[4]=42 vet[5]=52
f0   3: vet[0]=52 vet[1]=42 vet[2]=32 vet[3]=22 vet[4]=12 vet[5]=2
f1   4: i=0 j=13 *a=5 *b=5
main 5: i=70 j=5 v[1]=12
f1   6: i=0 j=23 *a=17 *b=17
main 7: i=3 j=4 w[4]=17 w[5]=70
main 8: 2.000000
main 9: 52 42 32
main 10: 22 17 70
```

`(nusp%100)/10 == 3`.

```
Digite o seu no. USP: 1234537
main 0: nusp=1234537
main 1: n=123453 k=3
f0   2: vet[0]=3 vet[1]=13 vet[2]=23 vet[3]=33 vet[4]=43 vet[5]=53
f0   3: vet[0]=53 vet[1]=43 vet[2]=33 vet[3]=23 vet[4]=13 vet[5]=3
f1   4: i=0 j=14 *a=5 *b=5
main 5: i=70 j=5 v[1]=13
f1   6: i=0 j=24 *a=18 *b=18
```

```
main 7: i=3 j=4 w[4]=18 w[5]=70
main 8: 2.000000
main 9: 53 43 33
main 10: 23 18 70
```

(nusp%100)/10 == 4.

```
Digite o seu no. USP: 1234547
main 0: nusp=1234547
main 1: n=123454 k=4
f0   2: vet[0]=4 vet[1]=14 vet[2]=24 vet[3]=34 vet[4]=44 vet[5]=54
f0   3: vet[0]=54 vet[1]=44 vet[2]=34 vet[3]=24 vet[4]=14 vet[5]=4
f1   4: i=0 j=15 *a=5 *b=5
main 5: i=70 j=5 v[1]=14
f1   6: i=0 j=25 *a=19 *b=19
main 7: i=3 j=4 w[4]=19 w[5]=70
main 8: 2.000000
main 9: 54 44 34
main 10: 24 19 70
```

(nusp%100)/10 == 5.

```
Digite o seu no. USP: 1234557
main 0: nusp=1234557
main 1: n=123455 k=5
f0   2: vet[0]=5 vet[1]=15 vet[2]=25 vet[3]=35 vet[4]=45 vet[5]=55
f0   3: vet[0]=55 vet[1]=45 vet[2]=35 vet[3]=25 vet[4]=15 vet[5]=5
f1   4: i=0 j=16 *a=5 *b=5
main 5: i=70 j=5 v[1]=15
f1   6: i=0 j=26 *a=20 *b=20
main 7: i=3 j=4 w[4]=20 w[5]=70
main 8: 2.000000
main 9: 55 45 35
main 10: 25 20 70
```

(nusp%100)/10 == 6.

```
Digite o seu no. USP: 1234567
main 0: nusp=1234567
main 1: n=123456 k=6
f0   2: vet[0]=6 vet[1]=16 vet[2]=26 vet[3]=36 vet[4]=46 vet[5]=56
f0   3: vet[0]=56 vet[1]=46 vet[2]=36 vet[3]=26 vet[4]=16 vet[5]=6
f1   4: i=0 j=17 *a=5 *b=5
main 5: i=70 j=5 v[1]=16
f1   6: i=0 j=27 *a=21 *b=21
main 7: i=3 j=4 w[4]=21 w[5]=70
main 8: 2.000000
main 9: 56 46 36
main 10: 26 21 70
```

(nusp%100)/10 == 7.

```
Digite o seu no. USP: 1234577
main 0: nusp=1234577
main 1: n=123457 k=7
f0   2: vet[0]=7 vet[1]=17 vet[2]=27 vet[3]=37 vet[4]=47 vet[5]=57
f0   3: vet[0]=57 vet[1]=47 vet[2]=37 vet[3]=27 vet[4]=17 vet[5]=7
f1   4: i=0 j=18 *a=5 *b=5
main 5: i=70 j=5 v[1]=17
f1   6: i=0 j=28 *a=22 *b=22
main 7: i=3 j=4 w[4]=22 w[5]=70
main 8: 2.000000
main 9: 57 47 37
main 10: 27 22 70
```

(nusp%100)/10 == 8.

```
Digite o seu no. USP: 1234587
main 0: nusp=1234587
main 1: n=123458 k=8
f0   2: vet[0]=8 vet[1]=18 vet[2]=28 vet[3]=38 vet[4]=48 vet[5]=58
f0   3: vet[0]=58 vet[1]=48 vet[2]=38 vet[3]=28 vet[4]=18 vet[5]=8
f1   4: i=0 j=19 *a=5 *b=5
main 5: i=70 j=5 v[1]=18
f1   6: i=0 j=29 *a=23 *b=23
main 7: i=3 j=4 w[4]=23 w[5]=70
main 8: 2.000000
main 9: 58 48 38
```

```
main 10: 28 23 70
```

`(nusp%100)/10 == 9`.

```
Digite o seu no. USP: 1234597
main 0: nusp=1234597
main 1: n=123459 k=9
f0   2: vet[0]=9 vet[1]=19 vet[2]=29 vet[3]=39 vet[4]=49 vet[5]=59
f0   3: vet[0]=59 vet[1]=49 vet[2]=39 vet[3]=29 vet[4]=19 vet[5]=9
f1   4: i=0 j=20 *a=5 *b=5
main 5: i=70 j=5 v[1]=19
f1   6: i=0 j=30 *a=24 *b=24
main 7: i=3 j=4 w[4]=24 w[5]=70
main 8: 2.000000
main 9: 59 49 39
main 10: 29 24 70
```

QUESTÃO 2

Se quer uma função que, recebendo um vetor de inteiros e um inteiro `n`, `n > 0`, devolvesse o maior valor entre as primeiras `n` posições do vetor. Assim, se o vetor fosse `[2,5,3,5,4,8,3]` e `n=5`, devolveria `5`, e se `n=7`, devolveria `8`. Para cada função sugerida abaixo, indique no quadro correspondente se está correta ou não, e, caso não, dê um exemplo com um vetor de **não mais que 6 elementos**, mostrando o que a função devolve e o que deveria devolver.

(a)

```
int maximo(int A[MAX], int n) {
  int i, j, max;
  max = A[0]; i = 1; j = n-1;
  while(i<=j){
    if (A[i] < A[j] && A[j] < max)
        max = A[j];
    if (A[i] > A[j] && A[i] > max)
        max = A[i];
    i++; j--;
  }
  return max;
}
```

SOLUÇÃO

```
PODE FALHAR.

   n = 2

   +---+---+---+---+---+---+
   | 1 | 2 |   |   |   |   |
   +---+---+---+---+---+---+
     0   1   2   3   4   5

   Devolve: 1       Deveria devolver: 2
```

(b)

```
int maximo(int A[MAX], int n) {
  int i, j, max;
  max = A[0]; i = 1; j = n-1;
  while(i<=j){
    if (A[i] < A[j] && A[j] > max)
        max = A[j];
    if (A[i] > A[j] && A[i] > max)
        max = A[i];
    i++; j--;
  }
  return max;
}
```

SOLUÇÃO

```
PODE FALHAR.

   n = 2

   +---+---+---+---+---+---+
```

```
   | 1 | 2 |   |   |   |   |
   +---+---+---+---+---+---+
     0   1   2   3   4   5

   Devolve: 1      Deveria devolver: 2
```

(c)

```
int maximo(int A[MAX], int n) {
  int max;
  max=0;
  while(n>0){
    if (A[n] > A[max])
      n = max;
    n--;
  }
  return A[max];
}
```

SOLUÇÃO

```
PODE FALHAR.

   n = 2

   +---+---+---+---+---+---+
   | 1 | 2 | ? |   |   |   |
   +---+---+---+---+---+---+
     0   1   2   3   4   5

   Devolve: 1      Deveria devolver: 2
```

(d)

```
int maximo(int A[MAX], int n) {
  int max;
  max=0;
  while(n>0){
    n--;
    if (A[n] < A[max])
      max = n;
  }
  return A[max];
}
```

SOLUÇÃO

```
PODE FALHAR.

   n = 2

   +---+---+---+---+---+---+
   | 1 | 2 |   |   |   |   |
   +---+---+---+---+---+---+
     0   1   2   3   4   5

   Devolve: 1      Deveria devolver: 2
```

(e)

```
int maximo(int A[MAX], int n) {
  int i, j;
  i=0; j=n-1;
  while(i<j){
    if(A[i] < A[j])
      i++;
    if(A[i] < A[j])
      j--;
  }
  return A[i];
}
```

SOLUÇÃO

```
PODE FALHAR.
```

```
n = 2

+---+---+---+---+---+---+
| 1 | 1 |   |   |   |   |
+---+---+---+---+---+---+
  0   1   2   3   4   5

Devolve: entra em 'loop'   Deveria devolver: 1
```

QUESTÃO 3

Esta questão consiste na implementação de três funções. Todas as funções são simplificações **muito grandes** de alguns trechos de código que você escreveu para o seu EP4. Os nomes das funções são `frequencia_linha`, `liste_possiveis` e `main`. No sudoku, a missão do jogador é preencher com números as **casas** de uma dada **grade** parcialmente preenchida como a mostrada na figura (a) abaixo.

```
    1 2 3   4 5 6   7 8 9            1 2 3   4 5 6   7 8 9
  +-------+-------+-------+        +-------+-------+-------+
1 | 5 . 1 | . . . | . 9 6 |      1 | 5 7 1 | 2 8 3 | 4 9 6 |
2 | . . . | . 9 . | . 5 . |      2 | 2 4 3 | 7 9 6 | 8 5 1 |
3 | . . . | . . 5 | 2 . 7 |      3 | 6 8 9 | 4 1 5 | 2 3 7 |
  +-------+-------+-------+        +-------+-------+-------+
4 | 4 9 . | 1 . . | . 7 . |      4 | 4 9 6 | 1 3 2 | 5 7 8 |
5 | . . . | . . 7 | . . . |      5 | 8 5 2 | 9 4 7 | 1 6 3 |
6 | 1 3 . | . . . | . 2 . |      6 | 1 3 7 | 5 6 8 | 9 2 4 |
  +-------+-------+-------+        +-------+-------+-------+
7 | 3 . 4 | . 5 9 | . . . |      7 | 3 1 4 | 6 5 9 | 7 8 2 |
8 | . 2 8 | . 7 1 | . 4 . |      8 | 9 2 8 | 3 7 1 | 6 4 5 |
9 | 7 6 5 | 8 2 . | . . . |      9 | 7 6 5 | 8 2 4 | 3 1 9 |
  +-------+-------+-------+        +-------+-------+-------+

          Figura (a)                       Figura (b)
```

Na grade da figura (a), as posições com um ponto ('.') indicam **casas vazias** e as posições que inicialmente possuem um número são ditas **casas fixas**. O objetivo do quebra-cabeças é inserir um número entre 1 e 9 em cada casa vazia a fim de obtermos uma grade em que cada número ocorra uma **única** vez em cada linha, em cada coluna, e em cada bloco. Por exemplo, a solução do sudoku da figura (a) é mostrada na figura (b).

```
                          1 2   3 4
                        +-----+-----+
   1                  1 | 1 . | . . |
  +---+               2 | . 2 | . . |
1 | . |   ordem 1       +-----+-----+   ordem 2
  +---                3 | . . | 3 . |
                      4 | 2 . | . 4 |
                        +-----+-----+
```

A **lista de possíveis** números para uma determinada casa vazia é formada pelos elementos que não ocorrem na linha, na coluna e no bloco a que a casa pertence. Se a lista de possíveis números para uma casa vazia tem apenas um único elemento, então esta casa é chamada de **casa forçada** e deve, portanto, ser preenchida com este elemento. No sudoku de ordem 2 acima, a posição `(3,1)` é forçada pois:

1. os números que não ocorrem na linha 3 são 1, 2 e **4**;
2. os números que não ocorrem na coluna 1 são 3 e **4**; e
3. os números que não ocorrem no bloco a que pertence `(3,1)` são 1, 3 e **4**.

Assim, o único número que não ocorre na linha, na coluna e no bloco da posição `(3,1)` é o número **4** que, portanto, deve ser inserido nesta posição.

No enunciado desta questão são usadas as declarações a seguir. Como no EP4, a grade de um sudoku de ordem `n` **deverá** ocupar as linhas e colunas de índices `1,. . .,`n^2 de uma matriz `grade`.

```
#define MAXN       10
#define MAXN2      MAXN*MAXN+1
#define VAZIA      0
```

item (a) Escreva uma função de protótipo

```
void frequencia_linha(int n, int grade[MAXN2][MAXN2], int lin, int frequencia[MAXN2]);
```

que recebe uma matriz `grade` representando um sudoku de ordem `n` e o índice `lin` de uma linha e, para `k=0,1, . . ,n`^{2}, devolve na variável `frequencia[k]` o número de ocorrências do número `k` na linha de índice `lin`.

SOLUÇÃO

```
/*
 * SOLUCAO
 *
 */
void frequencia_linha(int n, int grade[MAXN2][MAXN2],
          int lin, int frequencia[MAXN2]) {
   int n2 = n*n;
   int j;

   /* 1. inicialize vetor de frequencia com 0 */
   for (j = 0; j <= n2; j++)
     frequencia[j] = 0;

   /* 2. conte a frequencia de cada numero na linha lin */
   for (j = 1; j <= n2; j++)
       frequencia[grade[lin][j]]++;
}
```

Para escrever a função `liste_possiveis` pedida no item (b) você **deve** usar **três funções** descritas a seguir **sem escrevê-las**.

Suponha que lhe são dadas as funções de protótipo

```
void frequencia_linha(int n, int grade[MAXN2][MAXN2], int lin, int frequencia[MAXN2]);

void frequencia_coluna(int n, int grade[MAXN2][MAXN2], int col, int frequencia[MAXN2]);

void frequencia_bloco(int n, int grade[MAXN2][MAXN2], int lin, int col, int frequencia[MAXN2]);
```

As três funções recebem uma matriz `grade` representando um sudoku de ordem `n`. Para `k=0,1, . . ,n`^{2}:

1. `frequencia_lin` devolve na variável `frequencia[k]` o número de ocorrências do número `k` na **linha** índice `lin`;
2. `frequencia_col` devolve na variável `frequencia[k]` o número de ocorrências do número `k` na **coluna** de índice `col`;
3. `frequencia_bloco` devolve na variável `frequencia[k]` o número de ocorrências do número `k` no **bloco** a que pertence a posição `(lin,col)`

item (b) Escreva uma função de protótipo

```
int liste_possiveis(int n, int grade[MAXN2][MAXN2], int lin, int col, int lista[MAXN2]);
```

que recebe uma casa `grade[lin][col] == VAZIA` e para `k=1,2, . . . ,n`^{2} devolve na variável `lista[k]` o número 1, se o número `k` pode ser inserido na casa `grade[lin][col]`, em caso contrário `lista[k]` é devolvido com o número 0. Além disso, a função devolve através do `return` o número de possíveis valores para a casa `grade[lin][col]`.

A sua função `liste_possiveis` deve usar **obrigatoriamente** as funções

`frequencia_linha`, `frequencia_coluna` e `frequencia_bloco`.

Você pode usar a função `frequencia_linha` do item (a) **mesmo que não a tenha feito**.

SOLUÇÃO

```
/*
 * SOLUCAO
 *
 */
int liste_possiveis(int n, int grade[MAXN2][MAXN2],
                    int lin, int col, int lista[MAXN2]) {
  int n2 = n*n;
  int freq_lin[MAXN2];
  int freq_col[MAXN2];
  int freq_blo[MAXN2];
  int k;
  int no_possiveis;
```

```
  /* 1. conte a frequencia de cada numero na linha lin */
  frequencia_linha(n, grade, lin, freq_lin);

  /* 2. conte a frequencia de cada numero na coluna col */
  frequencia_coluna(n, grade, col, freq_col);

  /* 3. conte a frequencia de cada numero no bloco da posica (lin,col) */
  frequencia_bloco(n, grade, lin, col, freq_blo);

  /* 4 construa lista do numeros que podem ser inseridos em (lin,col) */
  no_possiveis = 0;
  for (k = 1; k <= n2; k++)
      if (freq_lin[k]==0 && freq_col[k]==0 && freq_blo[k]==0) {
        lista[k] = 1;
        no_possiveis++;
      }
      else lista[k] = 0;

  return no_possiveis;
}
```

Para escrever a função `main` pedida no item (c) você **deve** usar as seguinte funções **sem escrevê-las**. Suponha que lhe é dada uma função de protótipo

```
void carregue_grade(int *n, int grade[MAXN2][MAXN2]);
```

que lê um sodoku e devolve em `*n` a sua ordem. Além disso, o quebra-cabecas é armazenado nas linhas de 1 a n^2 e nas colunas de 1 a n^2 da matriz `grade`; a linha 0 e coluna 0 não são utilizadas.

Suponha ainda que lhe é dada uma função de protótipo

```
void escreva_grade_tela(int n, int grade[MAXN2][MAXN2]);
```

que imprime na tela uma `grade` de um sudoku de ordem `n`.

item (c) Escreva uma função `main` que lê e imprime um sudoku de ordem `n`, onde $0 < n \leq 10$, e lê do teclado:

1. um número inteiro `m`, $m > 0$; e
2. `m` posições de casas do sudoku.

O `main` deve, para cada uma dessas posições lidas, imprimir uma mensagem indicando se ela é ou não uma **casa forçada**. Além disso, se a posição for uma casa forçada, o programa **deve preencher** essa casa com o único valor possível e **imprimir** na tela a grade do sudoku.

A sua função `main` deve usar **obrigatoriamente** as funções

`carregue_grade`, `liste_possiveis` e `escreva_grade_tela`. Você pode usar a função `liste_possiveis` do item (b) **mesmo que não a tenha feito**.

A seguir vemos um exemplo de execução do programa onde $m = 4$:

```
carregue_grade: Digite o nome do arquivo com a grade: ordem2.ss
    1 2   3 4
  +-----+-----+
1 | 1 . | . . |
2 | . 2 | . . |
  +-----+-----+
3 | . . | 3 . |
4 | 2 . | . 4 |
  +-----+-----+
main: Digite o numero de posicoes: 4
main: Digite uma posicao: 1 1
main: casa (1,1) nao esta vazia.

main: Digite uma posicao: 2 1
main: casa (2,1) nao e' casa forcada.

main: Digite uma posicao: 3 2
main: casa (3,2) nao e' casa forcada.

main: Digite uma posicao: 3 1
main: casa (3,1) e' casa forcada, numero 4 inserido na posicao.
    1 2   3 4
  +-----+-----+
1 | 1 . | . . |
```

```
2 | . 2 | . . |
  +-----+-----+
3 | 4 . | 3 . |
4 | 2 . | . 4 |
  +-----+-----+
```

SOLUÇÃO

```
/*
 * SOLUCAO
 *
 */
int main() {

  int grade[MAXN2][MAXN2]; /* grade do sudoku */
  int n;             /* ordem da grade */
  int m;             /* numero de posicoes a serem lidas */
  int lin;           /* indice de uma linha da grade */
  int col;           /* indice de uma coluna da grade */
  int no_pos;        /* para receber valor do da funcao liste_possiveis */
  int lista[MAXN2]; /* lista de valores possiveis para uma dada posicao */
  int k;             /* usada para percorrer a lista de possiveis */
  int i;             /* contador de posicoes lidas */

  /* 1 carregue e escreva a grade do sudoku */
  carregue_grade(&n, grade);
  escreva_grade_tela(n,grade);

  /* 2 leia o numero de posicoes */
  printf("Digite o numero de posicoes: ");
  scanf("%d", &m);

  /* 3 verifique cada posicao para decidir se e casa forcada */
  for (i = 0; i < m; i++) {
    /* 3.1 leia a posicao */
    printf("Digite uma posicao: ");
    scanf("%d %d", &lin, &col);

    if (grade[lin][col] == VAZIA) {
      /* 3.2 construa a lista de possiveis numeros para (lin,col) */
      no_pos = liste_possiveis(n, grade, lin, col, lista);

      /* 3.3 verifique se e casa forcada */
      if (no_pos == 1) {
        for (k = 1; lista[k] != 1; k++);
        grade[lin][col] = k;

        printf("main: (%d,%d) e' casa forcada"
               " numero %d inserido na posicao.\n",
               lin, col, grade[lin][col]);
        escreva_grade_tela(n,grade);
      }
      else printf("main: (%d,%d) nao e' casa forcada.\n\n", lin, col);
    }
    else printf("main: (%d,%d) nao esta vazia.\n\n", lin, col);
  }

  return 0;
}
```

Last modified: Mon Jun 25 13:37:11 BRT 2007