Prova 2 (2021/01)

Valor: 27 pontos

Ao concordar em fazer esta prova, eu juro que seguirei o **código de honra**:

1. Não ajudarei e nem pedirei ajuda a ninguém durante o período de submissão do exame;
2. Não divulgarei qualquer informação sobre as minhas soluções durante o período de submissão do exame.

É normal não entender completamente uma questão. Se você tiver dúvidas sobre qualquer questão, envie uma mensagem ***PRIVADA*** no Teams ou um e-mail para olmo@dcc.ufmg.br com o assunto **[PROVA2]** e explique a sua dúvida para mim. Tentarei responder o mais rápido possível, mas a resposta pode demorar se a mensagem for enviada fora do horário comercial. **Não responderei mensagens enviadas através do sistema do Moodle!**

Sumário

Esta prova está dividida em três blocos. No primeiro bloco, composto pelas questões **1** a **4**, você vai implementar funções para preencher e localizar arranjos em X em uma matriz de inteiros, sendo necessário trabalhar com matrizes e com a struct `Xis`. No segundo bloco, composto pela questão **5**, você vai implementar uma função para contar o número de caracteres que se repetem em uma *string*. Por fim, no terceiro bloco, composto pela questão **6**, você vai implementar uma função para identificar posições relativas entre círculos e triângulos em um sistema de coordenadas idêntico ao usado pela biblioteca Allegro.

Importante: você pode criar quantas funções extras desejar. Isso é **altamente** recomendável para as questões **3** e **6**. Dividir é a melhor estratégia para conquistar!

Exercícios

Para todos os exercícios, considere a definição abaixo, que é fornecida para vocês na versão básica da prova:

```
#define MAX_TAM 100
```

Para as questões **2** a **4**, considere o tipo de dados **`Xis`** descrito abaixo:

```
typedef struct Xis {
     int linha, coluna, tam;
} Xis;
```

Essa estrutura armazena as informações sobre um "bloco" **`Xis`** que pode ser inserido e/ou detectado em uma matriz de inteiros **M**. Um "bloco" **`Xis`** é uma submatriz quadrada de **M** que oferece uma representação visual (e estrutural) de um **"X"** a partir das células da matriz **M**. Essa representação primeiramente é feita por células **diferentes de zero**, que compõem as linhas diagonais que formam o **X**. Essas diagonais (ou linhas do **X**) sempre têm o mesmo tamanho e se cortam na **coordenada central** do "bloco" **`Xis`**. Para compor a representação do **X**, todas as outras células fora das diagonais e dentro do "bloco" do **`Xis`** devem ser iguais a **zero**. Os campos `linha` e `coluna` guardam a **coordenada central** do **`Xis`**. O campo `tam` guarda o tamanho da submatriz de **M** que forma "bloco" **`Xis`**, que é também o número de células que uma diagonal de **`Xis`** possui.

É possível identificar um "bloco" **`Xis`** no **Exemplo 1** abaixo. Todas as células que compõem o "bloco" **`Xis`** estão coloridas de **verde**, enquanto todas as células que compõem as diagonais que representam o **X** estão com os valores coloridos de **vermelho**. Esse **`Xis`** é definido pela tupla **(2, 3, 3)**, ou seja, está localizado na `linha`=2 e `coluna`=3 (**coordenada central**), e tem tamanho `tam`=3. Sempre considere que um **`Xis`** possui um valor de `tam` ímpar.

0	0	0	0	0
0	0	-1	0	1
0	0	0	3	0
0	0	1	0	1
1	0	1	1	0

Exemplo 1

1) Implemente uma função de nome **matrizIdentidade** que recebe uma matriz **M** e o seu número **n** de linhas e colunas como parâmetros e a transforma em uma matriz Identidade. A **matriz identidade é uma matriz** quadrada de ordem **n**, em que os elementos que pertencem à diagonal principal são sempre iguais a **1** e os outros elementos que não pertencem à diagonal principal são iguais a zero. Caso **n** seja um valor inválido, a matriz **M** não deve ser modificada. Protótipo:

```
void matrizIdentidade(int M[][MAX_TAM], int n);
```

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Exemplo 2

Valor: **2 pontos**.

2) Implemente uma função de nome `insereXis` que recebe como parâmetros uma matriz quadrada **M**, o seu tamanho **n**, e um **Xis x**. Esta função deve alterar as células de **M** que correspondem ao bloco do **Xis x**. Em outras palavras, a função deve atribuir o valor **1** a todas as células da matriz que compõem o **Xis,** e **0** a todas as outras dentro dos limites do bloco que contém o **Xis**. Por exemplo, considerando **M** como a matriz identidade do exemplo do exercício anterior e **Xis** definido pela tupla **(1,2,3)**, então a sua função deve atribuir **1** às células `M[0][1]`, `M[0][3]`, `M[1][2]`, `M[2][1]` e `M[2][3]`, e **0** a todas as outras dentro dos limites do **Xis**: `M[0][2]`, `M[1][1]`, `M[1][3]` e `M[2][2]`. Veja na matriz abaixo como fica essa inserção.

1	1	0	1	0
0	0	1	0	0
0	1	0	1	0
0	0	0	1	0
0	0	0	0	1

Exemplo 3

Caso os campos do **Xis** não sejam válidos ou caso não seja possível inserir todo o **Xis** na matriz **M**, a sua função não deve modificar nenhuma célula da matriz e deve retornar **0**. Caso contrário, a função deve retornar **1**. Protótipo:

```
int insereXis(int M[][MAX_TAM], int n, Xis x);
```

Valor: **5 pontos**.

3) Implemente uma função de nome **maiorXis** que recebe como parâmetros uma matriz quadrada **M**, o seu tamanho **n**, e uma coordenada **i** e **j** de **M**, que representam uma linha e uma coluna dessa matriz, respectivamente. Esta função deve retornar as informações do maior **Xis** de **M** com coordenada central em (linha=**i** e coluna=**j**). Na matriz do **Exemplo 4**, abaixo, se **i**=1 e **j**=2, a sua função deve retornar o **Xis** composto pelos campos (`linha`=1, `coluna`=2, `tam`=3). Considerando essa mesma matriz, se **i**=2 e **j**=3, a sua função deve retornar o **Xis** composto pelos campos (`linha`=2, `coluna`=3, `tam`=3), uma vez que as células **(0,3)** e **(2,1)** contém valores diferentes de zero e impedem um Xis de tamanho **5** a partir da coordenada central **(2,3)**. Lembre que um **Xis** pode ter tamanho **1**. No **Exemplo 4**, se **i**=0 e **j**=5, a sua função deve retornar o **Xis** composto pelos campos (`linha`=0, `coluna`=5, `tam`=1), que é um **Xis** composto por uma única célula diferente de zero. Se não for possível formar um **Xis** a partir da coordenada central **(i,j)**, retorne o **Xis** composto pelos campos (`linha`=**i**, `coluna`=**j**, `tam`=0). Na matriz do **Exemplo 4**, se **i**=1 e **j**=1, a sua função deve retornar o **Xis** composto pelos campos (`linha`=1, `coluna`=1, `tam`=0). Lembre que para considerar um **Xis** válido é necessário que as diagonais contenham inteiros diferentes de 0 e que todas as outras células do "bloco" que formam o **Xis** sejam zero.

0	**1**	0	**1**	0	**-1**
0	0	**1**	0	**1**	0
0	**1**	0	**3**	0	0
0	0	**1**	0	**1**	0
1	**10**	1	1	0	**5**
0	0	0	0	0	0

Exemplo 4

Protótipo:

```
Xis maiorXis(int M[][MAX_TAM], int n, int i, int j);
```

Valor: **7 pontos**.

4) Implemente uma função **lerXis**, que recebe como parâmetro um **Xis por referência** e o tamanho **n** da matriz quadrada. O usuário deverá entrar com os valores de **linha**, **coluna** e

tamanho do **Xis** pelo teclado e nessa ordem e sua função deverá verificar se os valores são válidos considerando uma matriz de tamanho **n**. Caso sejam inválidos a função deverá solicitar um novo **Xis** até que tenha um valor válido como entrada. Ao obter um valor válido, retorne o número de tentativas que foram realizadas.

Protótipo:

```
int lerXis(Xis *x, int n);
```

Valor: **5 pontos**.

5) Implemente uma função de nome **numCharsRepetidos** que retorna o número de letras repetidas da *string* **str** que é passada como parâmetro. Uma letra repetida é uma que aparece pelo menos duas vezes na string. A sua função não deve fazer distinção entre letras maiúsculas e minúsculas. Considere como letras apenas caracteres com código ASCII entre 65 e 90 (maiúsculas) e 97 a 122 (minúsculas). Exemplo: se a string for "*Adoro programar em C!*", a sua função deve retornar **4**, visto que as letras **a, o, r** e **m** aparecem pelo menos duas vezes cada uma . As aspas não fazem parte da *string*.
Protótipo:

```
int numCharsRepetidos(char str[]);
```

Valor: **4 pontos**.

6) Para este exercício, considere os tipos de dados abaixo:

```
typedef struct Ponto {
      float x, y;
} Ponto;

typedef struct Circulo {
      Ponto centro;
      float raio;
} Circulo;

typedef struct Triangulo {
      Ponto sup, base_esq, base_dir;
} Triangulo;
```

Neste exercício, você deve implementar a função **estaContidoCirculoTriangulo**, que recebe um `Circulo` e um `Triangulo` como parâmetros e calcula se o triângulo está contido dentro do `Circulo`. Um triângulo está contido em um círculo se **1)** o seu centro está dentro do círculo e **2)** todos os seus pontos **não ultrapassam** os limites da circunferência do círculo. Se ele estiver contido, então a função deve retornar **1**. Caso contrário, a função retorna **0**. Um círculo é definido pelas coordenadas **x** e **y** do seu **centro** e também pelo **raio**. Um `Triangulo` é definido por três pares de coordenadas **x** e **y** referentes a seus vértices. Considere o mesmo sistema de coordenadas da biblioteca

Allegro, ou seja, o eixo **x** cresce da esquerda para a direita e o eixo **y** cresce de cima para baixo. Protótipo:

```
int estaContidoCirculoTriangulo(Circulo cir, Triangulo tri);
```

Valor: **4 pontos**.

Execução no VPL

Para conseguir compilar o seu programa, você deve implementar uma versão sintaticamente correta de todas as funções pedidas. Sugiro fortemente que faça isso antes de pensar nas soluções. Para o exercício **2**, por exemplo, você pode implementar a seguinte função:

```
int insereXis(int M[][MAX_TAM], int n, Xis x) { return 0; }
```

Para facilitar, a versão inicial desta prova já vem com essas versões básicas.

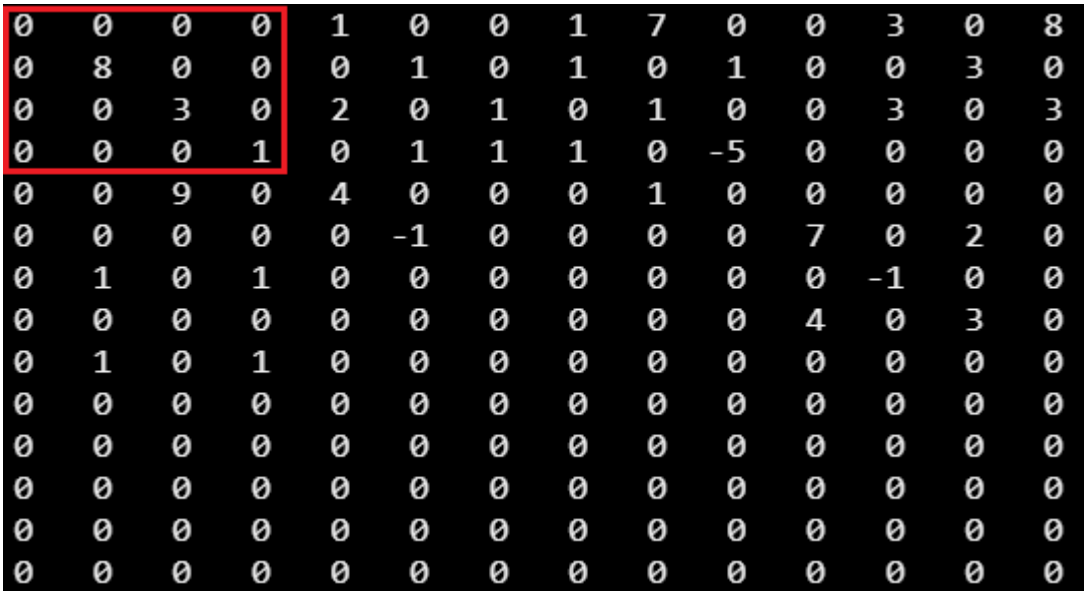
Uma forma de testar as suas soluções é através dos testes automáticos disponibilizados para vocês. Para executar os testes automáticos, clique no botão “avaliar” do VPL. Sugiro que faça isso antes de iniciar as soluções para poder visualizar todos os testes disponíveis para esta prova. Você pode avaliar o seu programa quantas vezes quiser. **Importante:** os testes **não** são completos e servem **apenas** para dar uma ideia **inicial** sobre a efetividade da questão.

Outra forma de testar as suas soluções no VPL é através do botão “**Executar**”. Para testar um exercício, inicie a execução no VPL e digite o número da função que gostaria de testar (**1**, **2**, **3**, **4, 5** ou **6**).

Para a função **1**, digite também o tamanho **n** da matriz.

Para a função **2**, digite também o tamanho **n** da matriz e a representação de um Xis (linha, coluna, tamanho). Exemplo: “**2 5 0 0 2**” testa a função **2**, cria uma matriz (de zeros) de tamanho **5** e insere um Xis de tamanho **2** na linha **0** e coluna **0**.

Para a função **3**, foi criada uma matriz de teste de tamanho **14** para você testar a sua função, que pode ser vista na figura abaixo:



Além de testar na matriz completa, você pode testar a sua função em submatrizes quadradas de tamanho **n** formadas a partir da da célula **(0,0)**. A submatriz de tamanho **n=4** é demarcada na figura por um retângulo vermelho. Assim, para testar a sua função digite também o tamanho **n** da matriz e a linha e a coluna da matriz em que você deseja procurar pelo maior Xis. Exemplo: "`3 4 1 1`" testa a função **3** na célula (1,1) da submatriz de tamanho **4**.

Para a função **4**, digite também o tamanho **n** da matriz e as entradas pedidas pela sua função **`lerXis`**. Exemplo:

```
4 10
3 3 3
```

Neste exemplo, uma matriz de tamanho **10** é criada e os valores linha=3, coluna=3 e tam=3 são lidos pela função **`lerXis`**.

Para a função **5**, digite também a string para ser avaliada. Exemplo:

```
5
É mentira! Eu odeio programar em C!!! :(((((
```

Para a função **6**, digite também as coordenadas do círculo e do triângulo. Exemplo:

```
6
0 0 10
-5 10 5 20 2 7
```

Nesse exemplo, são criados um círculo de raio **10** com centro na coordenada **(0,0)** e um triângulo com vértice superior na coordenada **(-5, 10)**, com vértice esquerdo da base na coordenada **(5, 20)** e com vértice direito da base na coordenada **(2, 7)**.

Por fim, você pode chamar a função `minha_main()` dando o inteiro **0** de entrada para o programa. Você pode implementar o que quiser nessa função, que serve para você testar qualquer situação particular que você desejar. **Essa função precisa estar no seu código (ainda que vazia) e não será avaliada nesta prova!**