MAC122 – Princípios de Desenvolvimento de Algoritmos

DCC - IME - USP

2º Semestre de 2013

Prof.: Dr. Paulo Miranda

Prova 01: (29/08/2013)

Nome:	
№ USP:	**Curso:**
Assinatura:	

Questão	Valor	Nota
1	2,0	
2	2,0	
3	1,5	
4	4,5	
Total	**10,0**	

Instruções:

- Escreva seu nome completo, № USP e assine com caneta;
- A prova é **individual** e **sem consulta;**
- A tentativa de cola implica em **ZERO;**
- Não é permitido o uso de computadores, calculadoras ou celulares;
- As folhas da prova não devem ser separadas;
- Rascunhos podem ser feitos no verso das folhas;
- Não é permitido o uso de outro rascunho além das folhas fornecidas;

<u>Questão 1</u> – (2,0 pts)

Faça uma função em C que construa dinamicamente um vetor contendo os índices das ocorrências de um dado caracter **`c`** em uma string **`texto`** fornecida como parâmetro. Exemplos:

Para **`texto = "cubanos chegam segunda-feira"`** e **`c = 'a'`** a função deve devolver o vetor de inteiros **`(3, 12, 21, 27)`.**

Para **`texto = "I love IME-USP"`** e **`c = 'I'`** a função deve devolver o vetor de inteiros **`(0, 7)`.**

A função deve devolver `NULL` caso o caracter buscado não esteja presente no texto.

Use o protótipo abaixo:

```
int *IndicesDoCaracter(char texto[], char c, int *nc);
```

onde nc aponta para uma variável que irá guardar o tamanho (número total de elementos) do vetor alocado.

<u>OBS</u>: Não é para usar as funções do <string.h>.
Por questões de simplicidade, assuma o padrão ASCII.

Questão 2 – (2,0 pts)

Para uma dada string de texto, faça uma função em C que separa as *tags* (etiquetas) do texto:
Tags são identificadas como trechos do texto compreendidos entre uma marca de início (caracter **`'<'`**) e outra de fim (caracter **`'>'`**).
Por exemplo, no texto **`"A etiqueta <html> indica ao navegador"`** temos a presença da tag **`"<html>"`**.

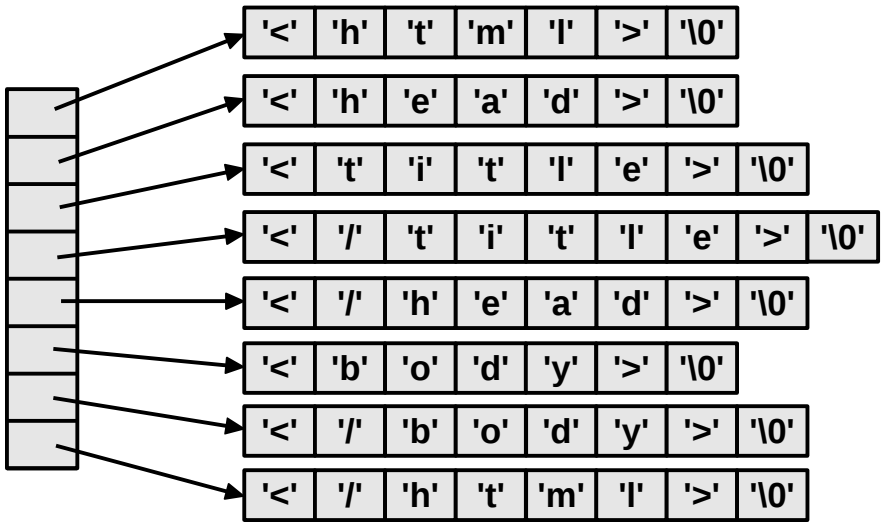
A função deve devolver o endereço de um vetor de apontadores criado dinamicamente com tamanho igual ao número de tags presentes no texto. Os apontadores desse vetor devem apontar para cópias das tags do texto. Cada cópia deve ser armazenada em um vetor alocado dinamicamente, usando a menor quantidade de memória possível.

Use o protótipo abaixo:

```
char **VetorDeTags(char texto[], int *ntags);
```

onde `ntags` aponta para uma variável que irá guardar o número de tags encontradas.

Exemplo:
Para o texto de entrada:
`"<html>\n<head>\n<title>MAC122</title>\n</head>\n<body>Prova</body></html>"`,
no final teremos os seguintes vetores alocados:



OBS: Assuma que os caracteres `'<'` e `'>'` sempre aparecem aos pares e nessa ordem (ou seja, para todo caracter `'<'` o próximo caracter de marcação é sempre um `'>'`). Por questões de simplicidade, assuma o padrão ASCII.

Dica: Você pode usar a função da questão anterior, mesmo que não a tenha implementado.

Questão 3 – (1,5 pts)

Implemente as seguintes funções auxiliares, que serão úteis na resolução da questão 4:

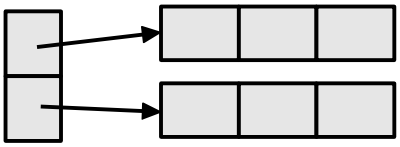
Item A:

Faça uma função em C que aloca dinamicamente memória para uma matriz de inteiros com m linhas e n colunas. A função deve alocar um vetor de apontadores e depois um vetor de inteiros para cada linha (vetor de vetores). O endereço do vetor de apontadores deve ser devolvido pela função.

Use o protótipo abaixo:

```
int **AlocaMatriz(int m, int n);
```

Exemplo: Para m = 2 e n = 3 teremos algo como:



Item B:

Faça uma função em C para liberar toda a memória de uma matriz com m linhas, alocada previamente pela função do item anterior.

Use o protótipo abaixo:

```
void LiberaMatriz(int **M, int m);
```

Item C:

Faça uma função em C que devolve o maior valor armazenado em uma matriz M, com m linhas e n colunas.

Use o protótipo abaixo:

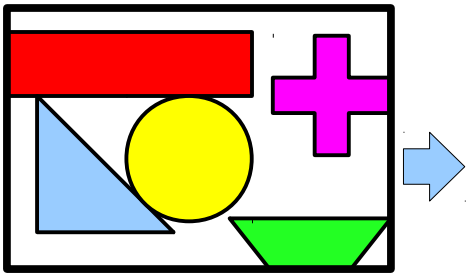
```
int MaiorValor(int **M, int m, int n);
```

Exemplo: Para a matriz abaixo a função deve devolver o valor 12.

$$\begin{bmatrix} 5 & 3 & 7 \\ 3 & 8 & 2 \\ 12 & 6 & 0 \\ 4 & 10 & 9 \end{bmatrix}$$

Questão 4 – (4,5 pts)

É dada uma matriz representando um desenho discreto aproximado de objetos geométricos. Cada valor da matriz, com m linhas e n colunas, representa um identificador do objeto ao qual o ponto da matriz pertence, e zero caso o ponto não pertença a nenhum objeto.



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 3 3 0 0 0
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 3 3 0 0 0
1 1 1 1 1 1 1 1 1 1 1 0 3 3 3 3 3 3 3 3
0 2 0 0 0 0 0 4 4 0 0 0 3 3 3 3 3 3 3 3
0 2 2 0 0 0 4 4 4 4 0 0 0 0 0 3 3 0 0 0
0 2 2 2 0 4 4 4 4 4 4 0 0 0 0 3 3 0 0 0
0 2 2 2 2 4 4 4 4 4 4 0 0 0 0 3 3 0 0 0
0 2 2 2 2 2 4 4 4 4 0 0 0 0 0 0 0 0 0 0
0 2 2 2 2 2 2 4 4 0 0 0 5 5 5 5 5 5 5 5
0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 5 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5 0 0
```

Item A:

Faça uma função que calcula o número de elementos de cada objeto (área dos objetos). Use o protótipo abaixo:

```
int *Area(int **M, int m, int n);
```

Os valores das áreas dos objetos devem ser armazenados em um vetor de inteiros, sendo o índice para cada área calculada dado pelo identificador do objeto correspondente.
O vetor deve ser alocado dinamicamente e a posição no índice zero (que não corresponde a nenhum objeto) deve, por convenção, ser usada para armazenar o número total de elementos do vetor (tamanho).
Ex: Para a matriz do exemplo acima teríamos o vetor `[6, 33, 21, 26, 24, 18]`

o vetor tem 6 elementos.

áreas dos objetos com índices de 1 a 5.

OBS: A função deve ser projetada para tratar qualquer matriz de entrada genérica, com um número arbitrário de objetos, e não somente 5 como no exemplo.

Item B:

Um elemento da matriz é considerado ponto de borda/contorno de um objeto, se ele pertence a um objeto, e pelo menos um de seus quatro vizinhos (cima, baixo, direita, esquerda) não pertence ao mesmo objeto.
Dada uma matriz de objetos (como no exemplo do item A), faça uma função que devolve uma segunda matriz (alocada dinamicamente com as mesmas dimensões da primeira), onde todos pontos de borda possuem valor 1, e zero caso contrário.

Use o protótipo abaixo:

```
int **Bordas(int **M, int m, int n);
```

Ex: Para a matriz anterior (do exemplo do item A) teríamos:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0
1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1
0 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 1 1
0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0
0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0
0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0
0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0
```

Item C:

Dada uma matriz de identificadores dos objetos (como no exemplo do item A), faça uma função que calcula o perímetro dos objetos. Considere o perímetro de um objeto como sendo seu número de pontos de borda/contorno.
Use o protótipo abaixo:

```
int *Perimetro(int **M, int m, int n);
```

Os valores dos perímetros dos objetos devem ser armazenados em um vetor de inteiros, sendo o índice para cada perímetro dado pelo identificador do objeto correspondente.
O vetor deve ser alocado dinamicamente e a posição no índice zero (que não corresponde a nenhum objeto) deve, por convenção, ser usada para armazenar o número total de elementos do vetor (tamanho).
Ex: Usando a matriz do item A como entrada, teríamos como resposta o seguinte vetor:

```
[6, 24, 15, 22, 12, 14]
```

Dica: Você pode usar as funções dos itens anteriores, mesmo que não as tenha implementado.