Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática (ICEI)
Engenharia de Computação / Engenharia de Software
Disciplina: Algoritmos e Estruturas de Dados I

Exercícios Funções

Exercício 1: Arranjos

Escreva um programa em C que tem uma função que recebe dois arranjos de números reais u e v e a dimensão n dos dois arranjos e que retorna o produto escalar de u e v. O produto escalar de dois arranjos é dado pela seguinte expressão:

$$u * v = u_0 * v_0 + v_1 * v_1 + ... + u_n * v_n.$$

Assuma que n é menor que o número máximo de elementos do arranjo (por exemplo, 100). Para testar, preencha cada vetor com um único valor.

Exercício 2: Fibonacci

A sequência de Fibonacci pode ser definida como:

fib(0) = 1
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2), para n>2

Implemente um programa que utilizando funções calcule a série de Fibonacci e armazene em um vetor cada elemento da série, de forma que a posição **0** armazene o termo **0**, a posição **1** o termo **1**, e assim por diante. Lembro que o termo **0** é o inteiro **1**. Seu programa deve receber do usuário um número entre **0** e **1000** e imprimir o termo correspondente ao número recebido. O usuário deve ser capaz de entrar com vários números interativamente em uma mesma execução. O programa termina quando o usuário entrar com um número **negativo** ou maior que **1000**.
Dica: se o tipo ***int*** for insuficiente para armazenar todos os elementos da série, use outro tipo de dados.

Exercícios Vetores

Exercício 3: Média dos elementos de um vetor***

Escreva uma função em C que recebe um vetor de números reais v e número de elementos n armazenados em v e que retorna a média dos n elementos armazenados em v. O vetor deve ser preenchido com números aleatórios através de uma outra função. Para gerar números aleatórios, use a função `drand48()` no Linux, ou a função `rand()` no Windows,

da biblioteca `stdlib.h`.

Exercício 4: Intercalação de vetores

Faça um programa que leia 2 vetores ***X*** e ***Y*** de 10 elementos, cada um. Intercale os elementos desses 2 vetores formando assim um novo vetor ***Z*** de 20 elementos, onde, nas posições ímpares de ***Z***, estejam os elementos de ***X*** e, nas posições pares, os elementos de ***Y***. Exemplo: Se ***X = 3,5,2,8,4*** e ***Y = 1,7,6,5,2*** então ***Z = 3,1,5,7,2,6,8,5,4,2***. Imprimir o vetor ***Z***.

Exercício 5: Inverso de um vetor

Faça um programa para ler um vetor X de n elementos e gerar um outro vetor com esses n elementos em ordem inversa. Exemplo: Se $X = 3, 5, 2, 8, 4$, deverá ser gerado um vetor $Y = 4, 8, 2, 5, 3$. O valor de n é lido pelo teclado.

Exercícios Strings

Exercício 6: Tamanho de uma string

Implemente um programa que leia um *string* e conte o número de caracteres. **Não** use a função `strlen`. Para ler uma *string* de até 127 caracteres do teclado, use o seguinte código:

```
char linha[128];
printf("digite uma linha:\n");
fgets(linha, 128, stdin);
```

Exercício 7: Conversão de caixa

Faça um programa que leia um *string* e modifique todos os caracteres minúsculos por caracteres maiúsculos.

Dica: os caracteres minúsculos tem o código ASCII entre 97 e 122 e os caracteres maiúsculos tem o código ASCII entre 65 e 90.

Exercício 8: Detector de palíndromos

Implemente código C para testar se uma *string* é um palíndromo. Um palíndromo é uma palavra idêntica quando lida de trás para frente, como "arara", "radar" e "reviver".

Exercício 9: Inversão de string

Escreva um programa para ler um texto do teclado e imprimir o inverso dele. Use o ponto

final para indicar o término do texto, ou seja, o usuário deve terminar o texto sempre com um ponto final. Leia um caractere do texto por vez usando a função getc como a seguir:

```
char c = getc(stdin);
```

Exercício 10: Abreviação de nome

Escreva um programa para ler um nome completo do teclado terminado com um ponto final (‘.’). Seu programa deve imprimir o mesmo na forma abreviada. Exemplo: o nome “Pedro Olmo Stancioli Vaz de Melo.” deve ser abreviado para “P.O.S.V.M.”. Note que o processo de abreviação deve ignorar palavras que começam com caracteres minúsculos. Considere que o usuário irá inserir apenas nomes válidos, sempre irá terminar o nome com o caractere ‘.’ e o nome não conterá acentos.

Exercícios Matrizes

Exercício 11: Sequencia Numérica

Neste exercício você deve fazer um programa para encontrar sequências de números iguais consecutivos, tanto na horizontal quanto na vertical, em uma matriz **m x n**. Depois, você deve substituir esses números por zeros e colocá-los nas primeiras linhas da matriz. Todo o programa será implementado a partir das questões a seguir.

1.1) Faça um programa para preencher uma matriz **m x n** com números aleatórios entre **1** e **k**. Os valores de **m**, **n** e **k** devem ser lidos do teclado. Como ainda não aprendemos alocação dinâmica de memória, crie uma matriz estaticamente com os limites superiores de **m** e **n**. Considere que **m** e **n** não podem ser maiores que **100**. Não permita que o usuário entre com valores inválidos para **m**, **n** e **k.**

Exemplo de uma matriz para m=5, n=4 e k=3:

3	**3**	**3**	**2**
3	**2**	**2**	**3**
1	**1**	**1**	**1**
2	**1**	**2**	**1**
2	**3**	**3**	**1**

1.2) Procure por sequências com pelo menos três números consecutivos iguais tanto nas linhas quanto nas colunas da matriz. Substitua todos os números que estão nessas sequências por **0**.

Depois de executar este procedimento na matriz exemplo do item anterior, ela deverá ficar assim:

0	0	0	2
3	2	2	3
0	0	0	0
2	1	2	0
2	3	3	0

1.3) Imprima na tela o número de zeros que a matriz possui depois do passo **1.2**.

Para a matriz do exemplo anterior, o seu código deve imprimir: 9

1.4) Altere a matriz colocando todos os zeros nas primeiras linhas das suas respectivas colunas. Preserve a ordem dos outros números dentro da coluna. Imprima a matriz final.

Para a matriz do exemplo anterior, o seu código deve imprimir a seguinte matriz:

0	0	0	0
0	0	0	0
3	2	2	0
2	1	2	2
2	3	3	3

1.5) Repita os procedimentos descritos nos itens 1.2, 1.3 e 1.4 até que a matriz final não tenha sequências de tamanho maior ou igual a 3 de números consecutivos maiores que zero.

Para a matriz do item anterior, o seu programa deve realizar as seguintes operações:

a) **Encontrar sequências de tamanho maior ou igual a três de números maiores que zero e substituir os números por zeros:**

0	0	0	0
0	0	0	0
3	2	2	0
2	1	2	2
2	0	0	0

b) **Colocar os zeros no topo da matriz:**

0	0	0	0
0	0	0	0
3	0	0	0
2	2	2	0
2	1	2	2

c) **Encontrar sequências de tamanho maior ou igual a três de números maiores que zero e substituir os números por zeros:**

0	0	0	0
0	0	0	0
3	0	0	0
0	0	0	0
2	1	2	2

d) **Colocar os zeros no topo da matriz:**

0	0	0	0
0	0	0	0
0	0	0	0
3	0	0	0
2	1	2	2

e) **Não há mais sequências de tamanho maior que 3 de números maiores que zero. Imprima a matriz final e termine o programa.**

Exercício 12: Alocação dinâmica de matrizes

Escreva um programa para ler um número inteiro ***n*** do teclado e criar dinamicamente uma matriz ***n x n*** de pontos flutuantes, atribuindo ***0.0*** a todas as suas posições. Uma matriz de pontos flutuantes de dimensões ***n x n*** é, na verdade, um vetor de ***n*** ponteiros para pontos flutuantes em que cada ponteiro deste vetor aponta para um vetor de ***n*** pontos flutuantes. Assim, para resolver esse exercício, primeiro aloque dinamicamente um vetor de ***n*** posições de ponteiros para pontos flutuantes. Depois, para cada posição ***i*** deste vetor, aloque um vetor de tamanho ***n*** de pontos flutuantes e atribua ***0.0*** a cada uma das suas posições. Por fim, imprima a matriz.

Exercícios Registros

Exercício 1

Neste exercício, você deve criar um protótipo de um sistema de batalha entre guerreiros de um jogo. Para isso, implemente os itens a seguir em um módulo separado chamado `jogo`.

1.1. Defina um novo tipo de dados chamado `Guerreiro` com os seguintes campos: `ataque` (inteiro), `defesa` (inteiro), `carisma` (inteiro), `pontos_vida` (inteiro) e `id_guerreiro` (inteiro).

1.2. Escreva uma função de nome `rolaDados` que simula a rolagem de três dados de seis faces tradicionais (1 a 6) e retorna a soma dessas rolagens. Note que somar os valores resultantes da rolagem de três dados de seis faces é diferente de rolar um dado que retorna um número entre 3 e 18.

1.3. Escreva um procedimento de nome `criaGuerreiro` que recebe um `Guerreiro` por passagem de parâmetro por **referência** e que atribui valores aos seus campos de batalha: `ataque, carisma` e `defesa`, **nessa ordem**. Os seus campos de batalha devem receber um valor inteiro da função `rolaDados`. Depois, atribua um valor para o campo `pontos_vida`, que deve receber a soma dos valores retornados por três execuções da função `rolaDados`. Assuma que o campo `id_guerreiro` já foi preenchido fora da função.

1.4. Escreva uma função de nome bonusCarisma que recebe um valor de carisma como parâmetro e retorna o bônus dado por esse valor de carisma. A tabela de bônus funciona da seguinte maneira:

- `Carisma` **18**: o guerreiro é extremamente carismático e tem todo o apoio da torcida, recebendo um bônus de **+3**.
- `Carisma` **16** e **17**: o guerreiro é muito carismático e tem o apoio de quase toda a torcida, recebendo um bônus de **+2**.
- `Carisma` **14** e **15**: o guerreiro é carismático e tem o apoio de alguns torcedores, recebendo um bônus de **+1**.
- `Carisma` **6** e **7**: o guerreiro é antipático, e tem alguma torcida contra ele, recebendo uma penalidade de **-1**.
- `Carisma` **4** e **5**: o guerreiro é muito antipático, e tem quase toda a torcida contra ele, recebendo uma penalidade de **-2**.
- `Carisma` **3**: o guerreiro é extremamente antipático, e tem toda a torcida contra ele, recebendo uma penalidade de **-3**.
- Para qualquer outro valor de carisma, a sua função deve retornar **0**.

1.5. Escreva um procedimento de nome `ataca` que recebe dois `Guerreiro`s por passagem de parâmetro por referência e simula um ataque do primeiro guerreiro no segundo. O ataque é dado da seguinte maneira:

a. O primeiro guerreiro rola três dados e soma os seus valores com o seu campo `ataque` e com o seu *bônus de carisma*. Essa soma é o valor do ***golpe*** do primeiro guerreiro.

b. O segundo guerreiro rola três dados e soma os seus valores com o seu campo `defesa` e com o seu *bônus de carisma*. Essa soma é o valor do ***escudo*** do segundo guerreiro.
c. Faça ***dano*** = ***golpe*** - ***escudo***. Se o ***dano*** for maior que zero, subtraia ***dano*** dos `pontos_vida` do segundo guerreiro. Ao subtrair o ***dano***, considere que o campo `pontos_vida` não pode ter valores menores que zero.

Desafios modo: BRABO

Alocação Dinâmica via Funções

Faça a alocação da matriz **Exercício 12** utilizando funções.

Abreviação de nome

Implementar um programa do **Exercício 10** considerando os possíveis erros do usuário:

- O número de espaços entre os nomes pode ser ilimitado. Ex: “John Doe”.
- Considerar que o usuário pode trocar letras maiúsculas por minúsculas e vice-versa. Ex: “John dOE”.
- Desconsiderar todas as palavras conectoras de nomes: de, da, do, das, dos.

Editor de textos

Implemente um programa para ler um texto de tamanho indefinido, armazená-lo em uma variável e imprimi-lo novamente na tela.

Passo a passo

1) Você deve ler caractere por caractere usando a função `getche()`. Para ler um caractere usando essa função, faça `char c = getche()`.

2) Todo o texto lido deve ser armazenado na memória a partir de alocação dinâmica. Crie um ponteiro para caractere (`char *texto`) para apontar para essa área de memória.

3) Antes de alocar memória para os caracteres, você deve armazenar temporariamente os caracteres lidos em um vetor de caracteres (`char buffer[BUFFER_TAM]`) de `BUFFER_TAM` posições. Para isso, conte os caracteres lidos usando uma variável (ex: `int contBuffer`) e armazene-os no vetor fazendo `buffer[contBuffer]=c`. Faça `#define BUFFER_TAM 5`.

4) Quando o vetor `buffer` estiver cheio, aloque dinamicamente outro espaço em memória e transfira todo o conteúdo do `buffer` para este espaço. Ao final, variável `texto` deverá receber o endereço para essa memória alocada:

```
texto = (char*)malloc((contBuffer)*sizeof(char));
```

Os detalhes desse processo são descritos a seguir. Sempre que vetor `buffer` estiver

cheio, aloque um novo espaço em memória para receber o conteúdo do `buffer` **mais** o conteúdo apontado pela variável `texto`. Crie um apontador de caracteres temporário de nome `char *textoaux` para apontar para esse espaço de memória. Transfira para esse espaço o conteúdo apontado por `texto` (caso exista) e, em seguida, o conteúdo de `buffer`.

5) Depois de fazer a transferência do item anterior, libere a memória apontada pelo apontador `texto` (que contém o texto desatualizado) e faça o apontador `texto` receber o endereço apontado por `textoaux` (que contém o texto atualizado). Dessa maneira, o apontador `texto` apontará para um espaço em memória que contém todo o texto digitado até o momento.

6) Este processo deve se repetir até que o caractere `'#'` seja digitado pelo usuário. Esse caractere não deve ser armazenado mas, ao invés dele, deve-se armazenar o caractere `'\0'`, delimitando o fim da *string*.

7) **Observação importante:** No Windows, se você apertar a tecla ENTER, os caracteres `'\r'` e `'\n'` serão enviados do teclado para a função `char c = getche()`. Dessa maneira, a variável `c` receberá somente `'\r'`, que retorna para o início da linha. Para fazer a quebra de linha corretamente, use o seguinte código após ler o caractere `c`:

```
if(c == '\r'){

      c = '\n';

      printf("\n");

}
```

Pseudo-código

Há outras maneiras de resolver este problema, algumas melhores que a apresentada abaixo!

```
#define BUFFER_TAM 5

Faça contBuffer = 0

faça {

      leia o caractere c do teclado

      se c == '\r', faça c = '\n' e imprima '\n' na tela

      se o buffer estiver vazio, faça buffer[contBuffer] = c e
contBuffer++

      se o buffer estiver cheio ou c == '#', faça {

            aloque memória para armazenar o conteúdo do buffer mais
o do texto até o momento armazenado

            faça textoaux apontar para essa área de memória

            transfira o conteúdo do texto e do buffer para essa área
de memória
```

```
            desaloque a área previamente alocada para o texto, caso
exista
            faça texto = textoaux
            zere o contador do buffer
      }
} enquanto (c != '#')
texto[countTotal-1] = '\0';
imprima o texto
desaloque a memoria alocada para o texto
```