

---

# Lista de Exercícios Práticos X

---

24 de maio de 2018

## 1 Alocação dinâmica de matrizes

Escreva um programa para ler um número inteiro  $n$  do teclado e criar dinamicamente uma matriz  $n \times n$  de pontos flutuantes, atribuindo 0.0 a todas as suas posições. Uma matriz de pontos flutuantes de dimensões  $n \times n$  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. DESAFIO: faça a alocação da matriz em uma função.

## 2 Editor de textos

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

### 2.1 Passo a passo

- Você deve ler caractere por caractere usando a função `getche()`. Para ler um caractere usando essa função, faça `char c = getche()`.
- 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.
- 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`.
- Na primeira vez que o vetor `buffer` estiver cheio, aloque dinamicamente um espaço em memória e transfira todo o conteúdo do `buffer` para este espaço. A variável `texto` deverá receber o endereço para essa memória alocada:  

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

  
Zere o contador `contBuffer` do `buffer`.
- Da segunda vez em diante que o vetor `buffer` estiver cheio, aloque um 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 as posições apontadas por `textoaux` o conteúdo apontado por `texto` e, em seguida, o conteúdo de `buffer`.
- Depois de fazer a transferência do item anterior, desaloque a memória apontada pelo apontador `texto` e faça o apontador `texto` receber o endereço apontado por `textoaux`. Dessa maneira, o apontador `texto` apontará para um espaço em memória que contém todo o texto digitado até o momento.
- 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*.
- **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 retornará 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");
}

```

## 2.2 Pseudo-código

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

```

#define BUFFER_TAM 5

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

```