Algoritmos e Estrutura de Dados - Laboratório 2
Tópicos: Structs, Funções e Ponteiros

No último laboratório, as atividades contemplaram um programa que manipula um vetor de números inteiros (digitado pelo usuário), de forma que todo número par era dividido por 2, substituindo o valor original no vetor. Foram implementadas 3 soluções:

Solução 1: Elaborar um programa monolítico, ou seja, sem uso de funções.

Solução 2: Refazer a solução do exercício 1, utilizando a função *manipula_pares*, que recebe o vetor (lido anteriormente na *main*) e altera os elementos pares. O vetor resultante é retornado para a *main*, que imprime o valor dos elementos.

Solução 3: Refaça a solução do exercício 1, utilizando além da função *manipula_pares*, a função *manipula_um_par.* Essa nova função recebe um único número inteiro, que é par, e divide por 2, alterando o valor do mesmo na memória. Ela é chamada dentro da *manipula_pares.* O vetor resultante é retornado para a *main*, que imprime o valor dos elementos.

Na aula de hoje daremos continuidade, fazendo uso de structs e vetor de structs, para reforçarmos os conceitos de passagem por valor e por referência, quando esses tipos compostos são utilizados.

Exercício 1. Refaça a solução do lab anterior (última versão), de forma que o vetor de inteiros seja substituído por um vetor de structs **dados** (suponha tamanho do vetor igual a 10). A struct **dados** deve conter um campo **nome** que armazena uma string (máximo de 5 caracteres) e outro campo **numero** que armazena um inteiro. Os números e nomes devem ser digitados pelo usuário. O programa também deve fazer uso das funções *manipula_pares e manipula_um_par,* conforme o exercício 3 do último lab. O vetor resultante é retornado para a *main*, que imprime o valor dos números resultantes (desprezar as strings na impressão). Obs: a função que manipula um par, recebe o valor inteiro, como foi definido no lab anterior.

Exercício 2. Refaça a solução do exercício 1, de forma que a função *manipula_um_par* receba a struct que contém um inteiro par no campo **numero**, e não apenas um número inteiro (como especificado anteriormente).

Exercício 3. Escreva um programa em C que declara e inicializa duas variáveis: um número inteiro **num** com o valor 5 e um número float **real** com o valor 4.9. Além disso, o programa deve usar duas variáveis **end_num** e **end_real** que armazenam os endereços das variáveis num e real. Essas variáveis devem ser declaradas como unsigned int (atenção! Não é para usar ponteiros aqui mas o simples conceito de uma variável que armazena endereço). O programa deve:

1º Bloco
- imprimir o valor de num
- imprimir o endereço de num em hexadecimal **sem usar** a variável end_num
- imprimir o endereço de num em hexadecimal **usando** a variável end_num
- imprimir o valor de end_num em hexadecimal
- repetir acima para real e end_real

2º Bloco
- alterar os valores das variáveis num e real, com uso de scanf, mas **sem usar** o operador de endereço (&)

3º Bloco
Repetir as impressões do 1º bloco

Exercício 4. Reescreva o programa anterior, usando ponteiros ao invés das variáveis **end_num** e **end_real**.

Exercício 5. (EXTRA) Refaça a solução do exercício 2, de forma que o programa manipule um vetor de structs **dados** como anteriomente, porém esse vetor deve ser alocado dinamicamente. O número de elementos do vetor deve ser informado pelo usuário.