UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade de Computação
Avenida João Naves de Ávila, 2121, Bloco 1B - Bairro Santa Mônica, Uberlândia/MG, CEP 38400-902
Telefone: +55 (34) 3239-4218 - www.facom.ufu.br - cocom@ufu.br

Bacharelado em Ciência da Computação
Bacharelado em Sistemas de Informação
Disciplina: Algoritmos e Estruturas de Dados 1 – AED1 [GBS024/GSI006]
Prof. Me. Claudiney R. Tinoco
Material baseado: Prof. Dr. Bruno Travençolo

2ª Lista de Exercícios – Revisão C: Ponteiros, Alocação Dinâmica e Funções

1) Existe um tipo de dado em linguagem C chamado "ponteiro para inteiro". Esse tipo de dado serve para armazenar um endereço de memória de uma variável inteira. Existem ponteiros para outros tipos de dados (e.g., float, double, char). Um ponteiro é uma variável como qualquer outra do programa – sua diferença é que ela não armazena um valor inteiro, real, caractere ou booleano. Ela serve para armazenar **endereços de memória** (que, no fundo, são valores inteiros sem sinal).

Para declarar uma variável do tipo ponteiro, use a seguinte sintaxe:
tipo_de_dado ***nome_da_variável;**

Note que, a única diferença na declaração de uma variável do tipo ponteiro é a adição de um símbolo ***. Assim, para declarar um ponteiro para inteiro devemos fazer assim:

```
int *p;
int *proximo;
int *anterior;
int *abacaxi;
```

Como um ponteiro serve para receber um endereço de memória, podemos fazer, por exemplo, a seguinte operação:

```
int a = 40; // cria uma variável do tipo inteiro, chamada a, e inicializa
            // com valor 40;
int *p;     // cria uma variável do tipo ponteiro para inteiro, chamada p, e o
            // conteúdo inicial é lixo;
p = &a;     // faz p receber o endereço de a. Dizemos que p aponta para a
```

Neste exemplo, dizemos que "p aponta para a" ou "p referencia a". Sabendo isso, continue o exercício 1 (não precisa criar um novo projeto) para:

(a) Copie o código acima e mostre o endereço da variável a de duas formas: uma usando **&a** e outra usando o ponteiro **p**. Os endereços devem ser os mesmos.
(b) Altere o valor da variável **a** usando o scanf sem usar o operador &.

2) Utilizando aritmética de ponteiros, mostre na tela o conteúdo da string char nome[] = “José Augusto”. Utilize o printf com %c e não %s.

3) Imprima o conteúdo de um vetor de double de 10 posições utilizando aritmética de ponteiros e SEM declarar variáveis do tipo ponteiro (ou seja, o nome do vetor terá que ser usado como o ponteiro).

4) Imprima o conteúdo de um vetor de int de 10 posições da última posição até a primeira utilizando aritmética de ponteiros e SEM declarar variáveis do tipo ponteiro (ou seja, o nome do vetor terá que ser usado como o ponteiro).

Alocação dinâmica: outra utilidade dos ponteiros é que eles permitem fazermos o que é chamado de alocação dinâmica. Isso significa que podemos reservar espaços de memória enquanto estamos executando o programa. Seria como criar variáveis com o programa executando. Por exemplo, considere um vetor que armazenará o preço de produtos. Não sabemos quantos produtos serão cadastradas enquanto estamos programando. Podemos fazer alocação dinâmica para resolver este problema. Implemente o programa abaixo, teste para diferentes valores de **n** e discuta o que significa o valor de **n** e de **sizeof(double)** no comando malloc(n*sizeof(double)).

```
double *produtos;
int n, i;

printf("Informe o número de produtos");
scanf("%d",&n);

// é necessário usar o comando malloc para alocar a memória
produtos = (double *) malloc(n*sizeof(double));

for (i = 0; i < n; i++){
    printf("Informe o valor do produto %d R$:",i+1);
    scanf("%lf", &produtos[i]);
}

printf("\nProdutos cadastrados\n");
for (i = 0; i < n; i++){
    printf("Produto %d  - R$: %f\n",i+1, produtos[i]);
}

// ao terminar de usar o vetor, devemos liberar a memória
free(produtos);
```

5) Crie um programa que:
   (a) Aloque dinamicamente um array de 5 números inteiros;
   (b) Peça para o usuário digitar os 5 números no espaço alocado;
   (c) Mostre na tela os 5 números;
   (d) Libere a memória alocada.

6) Faça um programa que leia n inteiros (definidos pelo usuário) armazenando-os em uma memória alocada dinamicamente. Em seguida, mostre quantos dos n números são pares e quantos são ímpares.

```
Quantos inteiros serão lidos: 3

1º inteiro: 4
2º inteiro: 6
3º inteiro: 7

São pares: 2 dos 3 inteiros lidos.
São impares: 1 dos 3 inteiros lidos.
```

7) Crie uma struct que armazene pontos inteiros (coordenadas x e y – valores inteiros). Crie dinamicamente um vetor de tamanho n (informado pelo usuário) e indique as coordenadas x e y de cada ponto. Ao final, mostrar todos os números digitados

Exemplo de saída:

```
Quantos pontos deseja digitar: 5
Entre com a coordenada x do ponto 1: 3
Entre com a coordenada y do ponto 1: 5
Entre com a coordenada x do ponto 2: 4
Entre com a coordenada y do ponto 2: 8
....
Entre com a coordenada x do ponto 5: 23
Entre com a coordenada y do ponto 5: 25

Pontos digitados: (3,5); (4,8); ... (23,25)
```

8) Faça uma função que calcule a área do retângulo definido por dois pontos.

Cabeçalho: area = calc_area(p1, p2)
Exemplo de saída:
A area do retangulo definido por (1, 4) e (4, 2) eh 6.

9) Faça um procedimento que multiplique o valor de um ponto por uma constante e altere o valor do ponto. Deve ser usada passagem por referência e retorno void.

Exemplo de saída:
Digite o ponto: 1,2
Digite a constante: 5
Resultado: (1,2) * 5 = (5,10)

10) Faça um procedimento chamado inc_dir, que faz o ponto andar uma unidade para leste, oeste, norte, sul (passar como referência e retorno void)

    Exemplo de chamada:
    inc_dir(p,'l'); // anda uma unidade para o leste (incrementa x)
    inc_dir(p,'o'); // anda uma unidade para o oeste (decrementa x)

11) Crie um procedimento que recebe um vetor de double como entrada e devolve o maior e o menor elemento do vetor. Mostre no programa principal o vetor, o maior e o menor elemento.

12) Alocação dinâmica em funções: sabemos que as variáveis locais de uma função (e também seus parâmetros) são alocadas na memória no momento da execução da função. Ao término da função, essas variáveis são destruídas da memória. Entretanto, quando fazemos uma alocação dinâmica dentro de uma função, o vetor alocado permanece na memória, mesmo após o término da função.

13) Crie uma função aloca_inteiro, que faz a alocação de um vetor inteiro de tamanho n e que inicialize os elementos desse vetor com o valor zero. Retorne o ponteiro para o vetor alocado. Imprima no programa principal o vetor alocado.

    Protótipo
    int* aloca_inteiro(int n);

    Chamada
    int *p;
    p = aloca_inteiro(10);
    imprime_vet(p,n);

14) Faça a função to_double, que recebe um vetor de inteiro e retorna um vetor com o mesmo conteúdo, mas convertido para double.

    Chamada
    vet_double = to_double(vet_int,n)