Lista de funções

Prof. Bruno Travençolo – FACOM-UFU

1) Faça um procedimento chamado DesenhaLinha. Ele deve desenhar uma linha na tela usando vários caracteres = (Ex: ========). No programa principal execute várias chamadas a esse procedimento (use um loop)

2) Altere o exercício anterior para que o procedimento aceite um argumento de quantos sinais de igual serão mostrados. Ex: DesenhaLinha(4) tem como saída ====; DesenhaLinha(10) tem como saída ==========. No programa principal execute várias chamadas a esse procedimento (use um loop)

3) Faça uma função para calcular o fatorial de um número. Use essa função para calcular o fatorial de um número que o usuário digitar. Colocar todas as funções E/S (entrada e saída) no programa principal (use também a função DesenhaLinha – esta pode conter comandos de saída fora do programa principal).

4) Faça uma função para calcular x^n. Não utilizar o operador pow (nem ^ - que não existe em C). No programa principal permita ao usuário informar o valor de x e de n (inteiro). Lembre que n pode assumir valor negativo.

5) Faça uma função para verificar se um número é um quadrado perfeito (retorne Verdadeiro caso seja).

6) Faça uma função para calcular o número neperiano usando uma série (ver abaixo). A função deve ter como parâmetro o número de termos que serão somados (note que, quanto maior o número, a resposta estará mais próxima do valor *e*). Observe que é preciso calcular o fatorial de vários números – para isso, utilize a função criada anteriormente. Utilize também o procedimento criado no exercício 2.

O número *e* pode ser representado e calculado por meio da utilização da série de Taylor para e^x quando x=1, como a soma da seguinte série infinita:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

Aqui *n*! representa o fatorial de *n*.

(fonte: Wikipédia)

7) Faça um procedimento “Troque”, que recebe duas variáveis reais A e B e troca o valor delas (i.e., A recebe o valor de B e B recebe o valor de A). Mostre no programa principal o resultado da troca

8) Faça uma função que altere um valor de um número real em x%. Se o valor de x for negativo ele deve ser decrementado, se for positivo aumente.

   Uso da função:
   x = incp(y,10);
   z = incp(y,-20);

9) Crie um procedimento idêntico ao exercício anterior, mas que mude o valor da variável passada por parâmetro (ou seja, o retorno deve ser void)

   Para os próximos exercícios crie uma struct chamada ponto, que armazena dois números reais que representam coordenadas cartesianas.

10) Faça uma função chamada imprime_ponto, que recebe uma struct do tipo ponto e mostra na tela o ponto no seguinte formato (ponto.x, ponto.y)

    Exemplo de chamada da função:
    imprime_ponto(p)

    Exemplo de saída resultante da função:
    (10,20)

11) Faca uma função que some dois pontos e retorne o resultado da soma. Mostre os 3 pontos usando a função imprime_ponto.

    ponto p1, p2, p3;
    p3= soma_ponto(p1,p2);

    Exemplo de saída:
    A soma de (1,2) com (3,4) é (4,6)

12) Crie um procedimento idêntico ao exercício anterior, mas que agora retorne void e coloque o resultado da soma no terceiro argumento da função.

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

Cabeçalho: area = calc_area(p1,p2)

Exemplo de saída:

A área do retângulo definido por (1,4) e (4,2) é 6

14) 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)

15) Faça um procedimento chamado inc_dir, que faz o ponto andar uma unidade para leste, oeste, norte, sul (passar como referencia 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)

    Exemplo de saída:
    (1,3) norte => (1,4)
    (2,4) leste => (1,4)

16) Crie um procedimento para andar na diagonal (sudeste, sudoeste, nordeste, noroeste). Use obrigatoriamente as funções do exercício anterior.

    ind_diag(p,"sudeste");
    ind_diag(p,"sudoeste");

17) Crie três funções chamadas imprime_vet_int, imprime_vet_double, imprime_vet_float que mostra o conteúdo de um vetor de inteiros, double, e float, respectivamente. Use essas funções sempre que precisar mostrar um vetor.
    Exemplo de Chamada:
    imprime_vet_int(vetori,n);
    imprime_vet_double(vetord,n);
    imprime_vet_float(vetorf,n);

18) 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

19) Implemente uma função que receba como parâmetro um vetor de n números reais (VET) de tamanho N e retorne quantos números negativos há a nesse vetor. Use o seguinte protótipo

    int negativos(float *vet, int N);

20) Faça uma função para copiar um vetor inteiro para outro vetor. Ambos vetores devem ter o mesmo tamanho. Mostre no programa principal os dois vetores.

    Chamada:
    copiarvet(vet_origem, vet_destino, n)

21) Faça uma função para multiplicar um vetor por um escalar (um número). Mostre, no programa principal, o vetor antes e depois da multiplicação.

multvet(vetor, n, escalar)

22) Faça uma função para transformar os números de um vetor de inteiros em seu valor absoluto (use a função abs (math.h))

    abs_vet(vet)

======= Alocação dinâmica com funções ==========

23) Use as funções criadas nos exercícios anteriores em um único programa, mas agora trabalhe com vetores alocados dinamicamente no programa principal. Lembre de liberar a memória depois de usar os vetores.

Exemplo de saída:

Entre com o tamanho do vetor: 4
Entre com o elemento 1: 50
Entre com o elemento 2: 20
Entre com o elemento 3: 10
Entre com o elemento 4: -30

Vetor: 50; 20; 10; -30
Maior: 50 Menor: -30
Número de negativos: 1
Vetor absoluto: 50; 20; 10; 30

Copiando para outro vetor (alocado dinamicamente)
Vetor original: 50; 20; 10; -30
Vetor copiado: 50; 20; 10; -30

Entre com o valor escalar para multiplicar o vetor original: 10

Vetor original: 500; 200; 100; -300

Vetor copiado: 50; 20; 10; -30

24) 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.

    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)

25) Faça uma função que copia um vetor de double para um outro vetor. Esse outro vetor é alocado dentro da função que faz a cópia. Mostre os dois vetores.

    Chamada:
    vet_destino = copiarvet(vet_origem, n)

26) 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)

27) Crie uma função para alocar dinamicamente um vetor de n pontos e inicialize esses pontos como (0,0).

28) Tente fazer uma função que copia um vetor de double para um outro vetor. Esse outro vetor é alocado dentro da função que faz a cópia. Mostre vet_destino no programa principal

    Chamada:
    copiarvet(vet_origem, vet_destino, n)

    Explique porque a função não funciona. Faça o mapa de memória (antes, durante, e depois da chamada da função). Qual seria a solução para uma função deste tipo?