Exercícios com funções para estudo:

O objetivo, dos exercícios é **treinar a passagem** de informações **pelo *return*** e **lista de parâmetros POR ENDEREÇO**. Muitas das funções não se justificariam num programa por executarem apenas uma operação simples.

1. Construir um programa que calcula o volume do cilindro. O seu programa deve fazer a leitura do raio da base circular e da altura do cilindro. Use o valor de PI = 3,14. Imprimir o valor do volume calculado.

   Volume é dado por: $\pi r^2 h$

   O programa **deverá** ter a seguinte divisão de módulos:

   a. função1: que recebe os valores: do raio da base circular e da altura do cilindro e calcula o volume do cilindro. O valor do volume calculado, deve retornar pelo *return*;
   Protótipo da função: float volume_cilindro (float raio, float alt);

   b. **main**: faz a leitura do raio da base circular e da altura do cilindro, chama a função, passando os **valores** lidos e imprime o valor do volume do cilindro, retornado da função.

2. Construir um programa que calcula o comprimento de um arco, com angulo central em graus. O seu programa deve fazer a leitura do raio da circunferência e do angulo em graus. Use o valor de PI = 3,14. Imprimir o valor calculado.

   Comprimento é dado por: $\frac{\alpha \pi r}{180}$

   O programa deverá ter a seguinte divisão de módulos:

   a. função1: que recebe os valores: do raio da circunferência e do angulo em graus e, calcula o comprimento de um arco. O valor do comprimento de um arco calculado, deve retornar pelo *return*;
   Protótipo da função: float comp_arco (float raio, float ang);

   b. main: declara as variáveis do problema, faz a leitura do raio da circunferência e do angulo em graus, chama a função, passando os ***valores*** lidos e, imprime o valor do comprimento de um arco, retornado da função.

3. Construir um programa para calcular a potência de um número, isto é, calcular: x^y (x elevado a y), onde x e y são parâmetros da função O programa deverá ter a seguinte divisão de módulos:
   a) **Função1:** recebe pela lista de parâmetros os **valores** de x e y e, usando obrigatoriamente, processo repetitivo (NÃO UTILIZAR A FUNÇÃO pow()), calcular o valor de x elevado à y. O resultado deve retornar pelo ***return***.
   PROTÓTIPO: ***float*** pot (*float* x, ***int*** y);

b) ***main()*** deverá ler os valores x e y, chamar a função que calcula a potência passando os ***valores*** lidos e, imprime o valor retornado da função.

4. Construir um programa que recebe um valor real representando a temperatura em *Fahrenheit* e calcula o respectivo valor em *Celsius* . O programa deverá ter a seguinte divisão de módulos:
   a) **Função1:** recebe pela lista de parâmetros o **valor** da temperatura em *Fahrenheit*, calcula o valor em *Celsius* . O valor em *Celsius* deverá retornar através do ***return.***
   PROTÓTIPO: ***float*** converte (*float* F);

   b) ***main()*** deverá ler da temperatura em *Fahrenheit*, chamar a função que calcula o valor em *Celsius* passando o ***valor*** lido e, imprime o valor retornado da função.

5. Construir um programa que faz a leitura de três números inteiros DIFERENTES e verifica qual deles é o maior. O programa deverá ter a seguinte divisão de funções – a divisão de módulos no exercício serve apenas para estudarmos passagem de parâmetros por enderço.

   a) **função1** que faz a leitura dos três números inteiros. Valida sessão diferentes. Se ocorrer repetição, repetir leitura ATÉ que os três digitados sejam diferentes. A comunicação das informações lidas deverá ser pela lista de parâmetros.
   PROTÓTIPO: ***void*** ler_tres (***int*** *a, ***int*** *b, ***int*** *c);
   b) **função2** que recebe os valores de três números inteiros (pela lista de parâmetros), verifica qual deles é o maior e retorna através do comando ***return*** o maior deles.
   PROTÓTIPO: ***int*** maior_tres (***int*** a, ***int*** b, ***int*** c);
   c) **função3** que recebe os valores dos três números lidos e do maior valor encontrado (todos pela lista de parâmetros) e imprime todas as informações.
   PROTÓTIPO: ***void*** imprime_quatro (***int*** a, ***int*** b, ***int*** c, ***int*** m);
   d) **função *main*()** define as variáveis do problema e chama as funções definidas.

6. Construir um programa que calcula a solução x e y de um sistema de equações lineares do tipo: ax + by = c
   dx + ey = f é dada por:

$$x = \frac{c*e - b*f}{a*e - b*d} \qquad y = \frac{a*f - c*d}{a*e - b*d}$$

   O programa deverá ter a seguinte divisão de funções:

a) **Função1:** que faz a leitura dos coeficientes (a, b, c, d, e, f). A comunicação das informações lidas deverá ser pela lista de parâmetros.
PROTÓTIPO: ***void*** ler_coeficientes (***int*** *a, ***int*** *b, ***int*** *c, ***int*** *d, ***int*** *e, ***int*** *f);

b) **Função2:** que recebe os valores dos coeficientes (pela lista de parâmetros) e calcula x e y. A comunicação do x e do y deverá ser pela lista de parâmetros. Testar se não resulta em divisão por zero. Se resultar em zero, devolver pelo return o valor **0** (zero). Se não resultar em divisão por zero, devolver pelo return o valor **1** (um);
PROTÓTIPO: ***int*** sistema (***int*** a, ***int*** b, ***int*** c, ***int*** d, ***int*** e, ***int*** f, ***float****x, ***float*** *y);

c) **Função3** que recebe os valores dos coeficientes lidos, os de x y e o valor de retorno do comando r*eturn* (todos pela lista de parâmetros) e imprime os coeficientes e de acordo com o valor do *return* da função anterior imprimir mensagem: "Sistema sem solucao" e não imprimir x e y ou imprimir a solução do sistema;
PROTÓTIPO: ***void*** imprime (***int*** a, ***int*** b, ***int*** c, ***int*** d, ***int*** e, ***int*** f, ***float*** x, ***float*** y, ***int*** flag);

d) a **função *main***() que define as variáveis do problema e chama as funções definidas.

7. Considere o mesmo problema anterior, porém com **OUTRA** forma de armazenamento dos coeficientes. Considere armazenar os coeficientes (a, b, c, d, e, f) em um vetor **com 6** posições, uma para cada um dos coeficientes. Exemplo: supor os valores dos coeficientes

a = 3, **b** = 2, **c** = 2, **d** = 3, **e** = 5, **f** = 1

3	2	2	3	5	1
[0]	[1]	[2]	[3]	[4]	[5]

Divisão de módulos:

a) **Função1:** que faz a leitura dos coeficientes e os armazena no vetor A comunicação das informações lidas deverá ser pela lista de parâmetros.
PROTÓTIPO: : ***void*** ler_coeficientes (***int*** v[ ]);

b) **Função2:** que recebe o endereço do vetor dos coeficientes e calcula x e y. A comunicação do x e do y deverá ser pela lista de parâmetros. Testar se não resulta em divisão por zero. Se resultar em zero, devolver pelo return o valor **0** (zero). Se não resultar em divisão por zero, devolver pelo return o valor **1** (um);
PROTÓTIPO: ***int*** sistema (***int*** v[ ], ***float****x, ***float*** *y);

c) **Função3** que recebe o endereço do vetor dos coeficientes, os valores de x y e o valor de retorno do comando r*eturn* (todos pela lista de parâmetros) e imprime os coeficientes e de acordo com o valor do *return* da função anterior imprimir mensagem: "Sistema sem solucao" e não imprimir x e y ou imprimir a solução do sistema;
PROTÓTIPO: ***void*** imprime (***int*** v[ ], ***float*** x, ***float*** y, ***int*** flag);

e) a **função *main***() que define as variáveis do problema e chama as funções definidas.

8. Construir um programa que faz a leitura de dois conjuntos de N números **reais**, armazena-os em dois arranjos unidimensionais X e Y, (ambos os arranjos terão N elementos) e cria um terceiro arranjo Z, também com N elementos, contendo em cada posição o maior entre os elementos de X e Y de mesma posição. O programa deverá ter a seguinte divisão de funções:
   - **função1** que faz a leitura de N e dos N números reais e os armazena num arranjo. A comunicação das informações lidas deverá ser pela lista de parâmetros (**Observe** que a função deverá ler valores para **um** arranjo);
     PROTÓTIPO: ***void*** ler_vetor(***float*** v[ ], ***int*** * n);
   - **função2** que recebe os valores de dois arranjos unidimensionais, ambos com N elementos, (pela lista de parâmetros) e faz a montagem do terceiro arranjo como descrito acima. A comunicação do terceiro arranjo deverá ser pela lista de parâmetros;

PROTÓTIPO: ***void*** monta_vetor (***float*** A[ ], ***float*** B[ ], ***float*** C[ ], ***int*** n);

- **função3** que recebe pela lista de parâmetros o valor de N e os valores do vetor e imprime esses valores;
  PROTÓTIPO: ***void*** imprime_vetor(***float*** v[ ], ***int*** n);
- ***main*()** que define as variáveis do problema e chama as funções definidas.
  Observe que para ler os dois vetores, a função1 é chamada DUAS vezes, uma para cada uma das variáveis arranjo. E, a função3, que imprime um vetor, deve ser chamada 3 vezes. Uma para cada um dos vetores: os 2 lidos e o criado com os elementos dos dois.

9. Construir um programa que faz a leitura de N números **inteiros**, armazena-os em um arranjo unidimensional X e monta um segundo arranjo unidimensional Y, também de N elementos, da seguinte forma:

$$Y_i = \begin{cases} x_i^2 & \text{se } xi < 0 \\ x_i & \text{se } xi = 0 \\ 2x_i & \text{se } xi > 0 \end{cases}$$

O programa deverá ter a seguinte divisão de funções:

- **função1** que faz a leitura do N e dos N números inteiros e os armazena no arranjo. A comunicação das informações lidas deverá ser pela lista de parâmetros.
  PROTÓTIPO: ***void*** ler_vetor(***float*** v[ ], ***int*** * n);
- uma **função2** que recebe os N valores de um arranjo unidimensional (pela lista de parâmetros) e constrói o segundo arranjo unidimensional, como definido acima. A comunicação do segundo arranjo deverá ser pela lista de parâmetros.
  PROTÓTIPO: ***void*** monta_vetor (***float*** A[ ], ***float*** B[ ], ***int*** n);
- uma **função3** que recebe pela lista de parâmetros valor de N e valores de um arranjo unidimensional de inteiros e imprime esses valores.
  PROTÓTIPO: ***void*** imprime_vetor(***float*** v[ ], ***int*** n);
- a função ***main*()** que define as variáveis do problema e chama as funções definidas. Imprime os dois vetores: o lido e o construído.

Cadeia de Caracteres (string)

10. Construir um programa que faz a leitura de uma **cadeia de caracteres** (**string**) contendo uma frase e conta quantas palavras há na frase. Imprime a frase lida e número de palavras contadas. Dividir o programa no seguinte conjunto de módulos:
    - **função1**: leitura de uma cadeia de caracteres (string);
      PROTÓTIPO: ***void*** ler_frase(***char*** f[ ]);
    - **função2**: recebe uma cadeia de caracteres (string) e conta a quantidade de palavras que há na frase. Devolver pela lista de parâmetros quantidade encontrada.
      PROTÓTIPO: ***int*** conta_palavras (***char*** f[ ]);
    - **função** main: define as variáveis do problema e chama as funções e imprime a frase lida e número de palavras retornado da função.

11. Construir um programa que faz a leitura de duas cadeias de caracteres (string) X e Y, com tamanhos diferentes e, monta uma terceira cadeia de caracteres (string) Z contendo os valores das duas strings lidas, de forma intercalada. Isto é, intercalar os caracteres das duas strings. Exemplo: supor duas strings:
    Str1: “casa amarela”
    Str2: “grande girassol”
    Resultado: cgarsaan daem agrierlaassol”

    Imprimir as strings: lidas e a calculada. Dividir o programa no seguinte conjunto de funções – *a divisão apenas para estudo – não justifica criar uma função só para ler uma string estática:*

    a) **função1**: leitura de uma cadeia de caracteres. (chamada duas vezes)
       PROTÓTIPO: ***void*** ler_frase(***char*** f[ ]);

    b) **função2**: recebe duas cadeias de caracteres e monta a terceira como descrito acima.
       PROTÓTIPO: ***void*** intercala(***char*** f1[ ], ***char*** f2[ ], ***char*** inter[ ]);

    c) **função** main: define as variáveis do problema e chama as funções. A função 1, chamar duas vezes.