==========
UFC - CK0215-T01-2019.1 - Exercícios da Aula 03 (25/02/2019)

ERRATA: conforme questionado por um aluno durante a aula, há um erro no tópico 5 da aula, quando usei o identificador "soma" tanto para uma das funções do programa quanto para uma das variáveis dessa função. A correção, claro, é usar identificadores diferentes.

Observação: https://en.cppreference.com/w/cpp/language/lookup
----

OBSERVAÇÃO: conforme informado em sala, é proibido fazer aritmética de ponteiros com ponteiros para void (não faz sentido, pois não se sabe o tamanho dos objetos apontados pelo ponteiro, e por isso não se sabe o tamanho dos saltos a serem realizados com o ponteiro). Tecnicamente, isso é especificado na seção 8.7 do padrão C++17, que exige, para a soma entre ponteiro e inteiro, que o ponteiro aponte para tipo completamente definido, ao passo que void tem tipo incompleto (§6.9.1).

QUESTÃO 1: é o exercício do tópico 6 da aula (era para classe, ficou para casa).

QUESTÃO 2: escreva um programa que leia inteiros do usuário, guardando-os numa tabela de dispersão que trate colisões por encadeamento externo

  https://en.wikipedia.org/wiki/Hash_table#Separate_chaining_with_linked_lists

e que use o método da divisão como função de dispersão. Em outras palavras, a sua tabela será um vetor T de "m" listas encadeadas (isto é, "m" ponteiros, cada um apontando para o primeiro nó de uma lista, ou então valendo nulo, se a lista estiver vazia). Além disso, quando o usuário digitar um número "n", esse número deverá ser armazenado no início da lista encadeada T[n % m]. O valor >= 1 de "m" deverá ser informado pelo usuário no início da execução do programa. Use alocação dinâmica para alocar T e os nós das listas encadeadas. Lembre de desalocar tudo ao final do programa.

A quantidade de números que serão digitados pelo usuário não é conhecida a priori. Cada número não-negativo digitado deverá ser armazenado na tabela de dispersão. O primeiro número negativo digitado indicará o fim da entrada, quando então o programa deve imprimir na tela os números que foram armazenados.

Observação: uma boa tabela de dispersão aumentaria o valor de "m" à medida em que mais números fossem armazenados, mas, por simplicidade, isso não é exigido neste exercício.

OBSERVAÇÃO: alocar memória e esquecer de desalocá-la é um erro típico, mas para o qual a orientação-a-objetos fornece uma grande ajuda, uma vez que é possível embutir comandos de desalocação nos destrutores das classes. C++ possui classes que tornam o uso da alocação dinâmica muito mais conveniente e seguro, chamadas de ponteiros espertos:

  https://en.cppreference.com/book/intro/smart_pointers

Vale a pena aprender e usar unique_ptr na prática:

  https://en.cppreference.com/w/cpp/memory/unique_ptr

==========