Exercícios sobre recursão

Prof. André Kishimoto

IMPORTANTE!

- A primeira coisa que você deve fazer para **começar** a resolver os exercícios abaixo é: ***ficar offline!***
- Deixe o celular e outras distrações de lado (incluindo abas abertas no navegador) e **foque** na solução dos exercícios. Você nem precisa usar o computador para resolver os exercícios! Ou melhor, só precisará do computador na hora de escrever o código; as soluções em si, os algoritmos, você pode resolver no papel.
- Se quiser, forme uma dupla ou trio e discuta os exercícios propostos.
- Não acesse Google, Stack Overflow, ChatGPT, etc. O objetivo com os exercícios não é simplesmente apresentar uma resposta correta – isso é apenas uma consequência do **principal objetivo**: *"quebrar a cabeça"* (**pensar**) sobre como resolver os problemas apresentados e entender recursão (e, no processo, **detectar e tirar dúvidas** sobre o assunto).
- Colocando de outra forma: ao finalizar os exercícios, **reflita** e responda para si mesmo(a): *"Eu realmente sei resolver os exercícios, ainda que nem todos os detalhes sobre recursão (e tópicos associados) estejam totalmente claros para mim? Ou eu simplesmente copiei a resposta de algum lugar, só para enganar o professor? Se fiz isso, será que não estou enganando a mim mesmo(a)?"*
- **Lembre-se:** em uma avaliação, você precisa demonstrar que sabe **pensar e resolver problemas**, e nem sempre terá Google/ChatGPT/etc. para consultar. E avaliação não significa apenas prova de faculdade, pode ser um teste técnico em uma entrevista de emprego/estágio, uma prova de concurso...

Para todas as questões a seguir, implemente sua solução usando a linguagem C.

Q1. Escreva uma função recursiva `potencia(x, n)` que calcula e retorna o resultado de x^n (`x` e `n` são números inteiros).

Q2. Escreva uma função recursiva `soma_digitos(n)` que calcula e retorna a soma dos dígitos de um número inteiro `n`. Por exemplo, `soma_digitos(123)` deve retornar o valor `6` (1 + 2 + 3 = 6).

Q3. Escreva uma função recursiva `mul(a, b)` que calcula e retorna a multiplicação de `a` com `b` (números inteiros), sem usar o operador de multiplicação. Garanta que o cálculo seja realizado com a menor quantidade possível de chamadas recursivas.

Q4. Escreva uma função recursiva `mdc(a, b)` que calcula e retorna o máximo divisor comum (MDC) dos números inteiros `a` e `b`. O resultado do MDC de dois números inteiros é o maior valor numérico inteiro que divide os dois números sem deixar resto, simultaneamente.

Dica: O algoritmo de Euclides é uma técnica recursiva para encontrar o MDC de dois números inteiros e é baseado nas seguintes propriedades:
- MDC(***a***, 0) = ***a***
- MDC(0, ***b***) = ***b***
- MDC(***a***, ***b***) = MDC(***b***, ***r***), sendo ***r*** o resto da divisão de ***a*** por ***b***.

Q5. Escreva uma função recursiva `soma_array(arr[], tam)` que calcula e retorna a soma de todos os elementos de um *array*. Considere que `arr[]` é um *array* de números decimais (`float` ou `double`).

Q6. Escreva uma função recursiva `divisiveis(arr[], tam, x)` que retorna `true` se todos os números inteiros do *array* `arr[]` são divisíveis pelo inteiro `x` ou retorna `false`, caso contrário.

Q7. Altere a função recursiva `divisiveis()` da **Q6** para que, ao invés de retornar `true`/`false`, a função retorne a quantidade de números em `arr[]` que são divisíveis por `x`. **Atenção!** A sua solução não deve usar variáveis globais.

Q8. Altere a função recursiva `divisiveis()` da **Q7** para que o retorno da função seja a somatória dos números de `arr[]` que são divisíveis por `x`. **Atenção!** A sua solução não deve usar variáveis globais.

Dica para Q5-Q8: Quando um *array* é passado como argumento de uma função, o que realmente é passado para a função é o endereço de memória do primeiro elemento do *array* (ponteiro). Assim, podemos passar como argumento da função um *array* que começa em uma posição específica (não necessariamente na posição zero do primeiro elemento do *array*).

Para fazer isso, devemos somar um número inteiro `pos` com o endereço de memória do primeiro elemento do *array*, sendo `pos` a posição do elemento inicial que queremos passar como argumento da função. Dentro da função, esse `pos` será considerado o índice zero do *array*.

O código a seguir ilustra o que foi descrito nos parágrafos anteriores:

```c
#include <stdio.h>

void dica(int arr[])
{
  // Observe que estamos acessando arr[0] dentro da função dica(), que pode não
  // ser a posição zero do array passado como argumento.
  printf("valor na posição 0 do array: %d\n", arr[0]);
}

int main()
{
  int num[] = { 101, 99, -1, 2023, 404, 123 };
  dica(num + 1); // Exibe 99 (já que num + 1 aponta para o 2º elemento do array).
  dica(num + 3); // Exibe 2023.
  dica(num + 6); // CUIDADO! Acesso fora dos limites do array.
}
```

Q9. Escreva uma função recursiva `fib(n)` que calcula e retorna o `n`-ésimo termo da sequência de Fibonacci. Na sequência de Fibonacci, os dois primeiros termos são o número 1 e os próximos termos são definidos pela soma dos dois termos anteriores: { 1, 1, 2, 3, 5, 8, 13, 21, 34, ... }.

Q9.1. O que acontece quando `fib()` é chamado com `n >= 47`? O código precisa ser alterado? Se sim, o que deve ser alterado no código?

Q10. Dependendo de como você implementou a função recursiva `fib()` da **Q9**, talvez você tenha percebido que a função começa a demorar para calcular a resposta conforme `n` aumenta (por exemplo, `n=45`). Por que isso acontece?

Altere a função recursiva `fib()` da **Q9** para que a função realize o cálculo de maneira mais rápida. **Atenção!** A sua solução não deve usar variáveis globais.