ED1 2025/1: Lista de Exercícios do Cap. 4
Estrutura de Dados I
Professor(es): Abrantes Araújo Silva Filho   Entrega: 2025-04-01 23:59

Exercícios referentes ao Capítulo 4: Introdução à Recursão

1. Exercícios de programação:

   (a) Ao contrário de muitas linguagens, C não possui um operador primário para calcular a potência de algum número, por exemplo, calcular n^k. Como uma solução parcial para o problema, defina a função recursiva cuja declaração é

   ```
   int potencia (int n, int k);
   ```

   e que calcula n^k. Talvez seja útil você se lembrar da propriedade matemática de que:

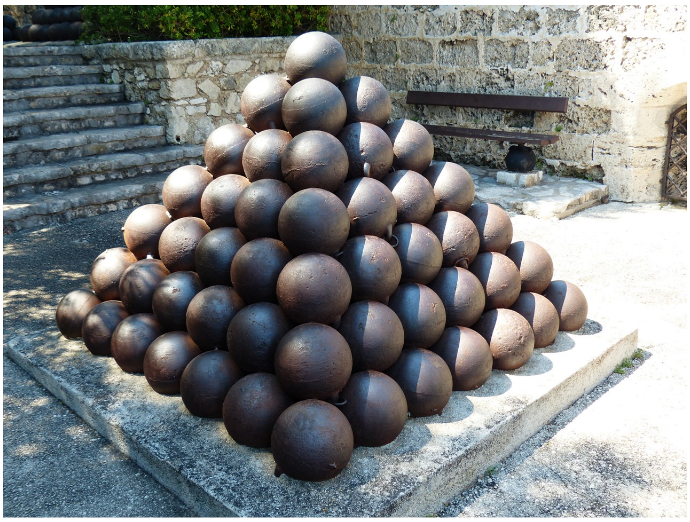
   $$n^k = \begin{cases} 1 & \text{se } k = 0 \\ n \times n^{k-1} & \text{caso contrário} \end{cases} \tag{1}$$

   (b) O **maior divisor comum** (mdc) de dois números inteiros não negativos é o maior número inteiro que divide uniformemente os outros dois números. Em 300 a.C. o matemático grego Euclides descobriu que o mdc de x e y poderia ser obtido da maneira como se segue:

   - Se x for uniformemente divisível por y, então y é o mdc;
   - Caso contrário, o mdc entre x e y será sempre igual ao mdc entre y e o resto da divisão de x por y.

   Utilize o método de Euclides para definir a função "`int gcd (int x, int y);`", que calcula o maior divisor comum entre x e y.

   (c) Objetos esféricos, como bolas de canhão, podem ser empilhadas em uma pirâmide com 1 bola no topo, sobre uma base quadrada composta de 4 bolas, sobre uma base quadrada composta de 9 bolas, e assim por diante. Escreva uma função recursiva chamada de "`bolas_de_canhao`" que recebe como argumento a altura a da pirâmide e retorna o número de bolas de canhão que essa pirâmide contém.



Fonte: Hans, no Pixabay

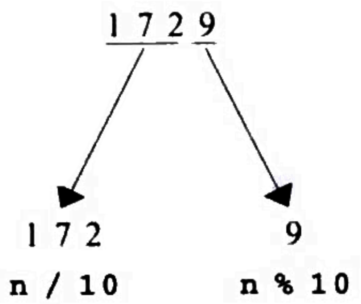
(d) Para cada uma das duas implementações da função "`fib(n)`" que estudamos nesse capítulo (uma primeira implementação que não era eficiente e uma segunda implementação mais eficiente), escreva funções recursivas chamadas de "`contagem_fib1`" e "`contagem_fib2`" que conta o número de chamadas recursivas feitas durante a execução de cada função na obtenção de um determinado número de Fibonacci. Escreva um programa que utiliza essas funções para mostar uma tabela com o número de chamadas recursivas que cada algoritmo faz para diversos valores de n, conforme ilustrado abaixo:

```
Este programa compara a performance de dois
algoritmos para o cálculo da Seqüência de Fibonacci:

Número de chamadas recursivas:
  N    Fib1    Fib2
 --    ----    ----
  0       1       2
  1       1       2
  2       3       3
  3       5       4
  4       9       5
  5      15       6
  6      25       7
  7      41       8
  8      67       9
  9     109      10
 10     177      11
 11     287      12
 12     465      13
 13     753      14
 14    1219      15
 15    1973      16
```

(e) Defina a função recursiva "`int somar_digitos (int n);`", que recebe um número inteiro não negativo e retorna a soma de seus dígitos. Por exemplo: chamar a função com o argumento 1729 deve retornar 19, pois é igual à $1 + 7 + 2 + 9$.

A implementação recursiva dessa função se deve ao fato de que é muito fácil quebrar um inteiro em seus dígitos componentes usando a divisão e o resto da divisão do número por 10. Por exemplo, para o inteiro 1729, você pode dividir esse número em duas partes como se segue:



Cada um dos inteiros resultantes é estritamente menor do que o original e representa um caso mais simples.

(f) A **raiz digital** de um número inteiro não negativo n é definida como o resultado de somar repetidamente seus dígitos, até que um único dígito seja obtido. Por exemplo: a raiz digital de 1729 pode ser obtida seguindo-se os passos a seguir:

1: $1 + 7 + 2 + 9 = 19$

2: $1 + 9 = 10$

3: $1 + 0 = 1$

Como o total ao final do terceiro passo foi o único dígito "1", esse é o valor da raiz digital.

Escreva a função "`int raiz_digital (int n);`", que recebe como argumento um número inteiro não negativo e retorna sua raiz digital. Embora seja fácil implementar a função `raiz_digital` usando a função `somar_digitos` do exercício anterior e um loop `while`, lembre-se de que você não pode utilizar nenhum loop em seus programas e, portanto, parte do desafio neste problema é escrever esta função em termos totalmente recursivos, sem usar nenhuma estrutura de repetição em seu programa.

(g) A função matemática $C(n,k)$ é usualmente definida em termos de fatoriais, como se segue:

$$C(n,k) = \frac{n!}{k! \times (n-k)!} \tag{2}$$

Os valores de $C(n,k)$ também podem ser arranjados geometricamente para formar um triângulo no qual n aumenta à medida em que você desce o triângulo, e k aumenta a medida em que você percorre o triângulo da esquerda para a direita. A estrutura resultante é chamada de **Triângulo de Pascal**, em homenagem ao matemático francês Blaise Pascal, e é construída da seguinte maneira:

```
                C(0,0)
            C(1,0) C(1,1)
        C(2,0) C(2,1) C(2,2)
    C(3,0) C(3,1) C(3,2) C(3,3)
C(4,0) C(4,1) C(4,2) C(4,3) C(4,4)
```

O Triânculo de Pascal tem uma propriedade interessante que nos mostra que cada número é a soma dos dos números imediatamente superiores, exceto ao longo dos lados esquerdo e direito, cujos números são sempre 1. Considere, por exemplo o número 20 no Triângulo de Pascal abaixo:

```
                  1
               1     1
            1     2     1
         1     3     3     1
      1     4     6     4     1
   1     5    10    10     5     1
1     6    15    20    15     6     1
```

Esse número 20 corresponde à $C(6,2)$, que é a soma das duas entradas imediatamente superiores (5 e 10). Use esse relacionamento entre as entradas em um Triângulo de Pascal para escrever uma implementação recursiva de $C(n,k)$ que não utiliza nenhum loop, nenhuma multiplicação e nenhuma chamada à nenhuma função fatorial (seja recursiva ou iterativa).

(h) Escreva uma função recursiva que recebe uma string como argumento e retorna o reverso dessa string. O protótipo para essa função deve ser

```
string reverter (string str);
```

e a sentença

```
printf("%s\n", reverter("programa"));
```

deve retornar

```
amargorp
```

(i) A biblioteca `strlib.h` contém a função `IntegerToString`, que é implementada usando a função `sprintf`, como se segue:

```
#define MaxDigits 30

string IntegerToString(int n)
{
    char buffer[MaxDigits];

    sprintf(buffer, "%d", n);
    return (CopyString(buffer));
}
```

Esta implementação, entretanto, esconde todo o trabalho dentro da função `sprintf` e não fornece nenhuma explicação sobre como o computador realmente executa o processo de converter um número inteiro em sua representação como string. Ocorre que a maneira mais fácil para implementar essa função é usar a decomposição recursiva de um inteiro conforme você vez no exercício "1. (e)" anterior. Reescreva a implementação da função `IntegerToString` para que ela opere de maneira recursiva. Sua função pode chamar qualquer outra função da biblioteca `strlib.h`, exceto as funções `IntegerToString` e `RealToString`, mas também não pode utilizar outras funções de outras bibliotecas tais como a `sprintf`. Lembre-se: sua função deve operar recursivamente e não pode usar nenhuma estrutura iterativa.

2. Exercícios conceituais:

(a) Defina os termos **recursivo** e **iterativo**. É possível que uma função utilize as duas abordagens?

(b) Explique o que é o paradigma recursivo.

(c) Quais as duas propriedades que um problema deve ter para que a recursão seja uma possibilidade como estratégia de solução?

(d) Costumamos dizer que a recursão é um exemplo de estratégia **dividir para conquistar**. Explique.

(e) O que queremos dizer por **salto de fé recursivo**? Por que esse conceito é importante para você que é programador?

(f) Durante nosso estudo do fatorial recursivo nós fizemos o rastreamento de todo o processo das chamadas, mostrando todos os *stack frames* empilhados, as chamadas feitas e os retornos executados para o cálculo do fatorial de 4. Usando esse mesmo modelo de rastreamento, faça a demonstração detalhada do rastreamente do cálculo de `fib(4)`, desenhando cada *stack frame* que é criado ao longo do processo.

(g) Modifique o problema dos coelhos de Fibonacci introduzindo uma regra adicional: cada par de coelhos pára de reproduzir após dar à luz 3 ninhadas. Como essa regra afetaria a relação de recorrência? Que alterações você precisaria fazer aos casos simples?

(h) O que é uma função *wrapper* (invólucro)? por que elas são úteis na impementação de funções recursivas?

(i) O que é recursão mútua?